目　录

第5章 进化计算初步

第6章 遗传算法与数值优化

第7章 遗传算法与组合最优化

第8章 遗传算法与机器学习

第9章 遗传算法与智能控制

第10章 遗传算法与人工生命

第1章　遗传算法简介

遗传算法(Genetic Algorithms, GA)研究的历史比较短,20世纪60年代末期到70年代初期,主要由美国Michigan大学的John Holland与其同事、学生们研究形成了一个较完整的理论和方法,从试图解释自然系统中生物的复杂适应过程入手,模拟生物进化的机制来构造人工系统的模型。随后经过20余年的发展,取得了丰硕的应用成果和理论研究的进展,特别是近年来世界范围形成的进化计算热潮,计算智能已作为人工智能研究的一个重要方向,以及后来的人工生命研究兴起,使遗传算法受到广泛的关注。从1985年在美国卡耐基·梅隆大学召开的第一届国际遗传算法会议(International Conference on Genetic Algorithms: ICGA'85),到1997年5月IEEE的Transactions on Evolutionary Computation创刊,遗传算法作为具有系统优化、适应和学习的高性能计算和建模方法的研究渐趋成熟。本章在介绍遗传算法的产生和发展历史之后,概述了遗传算法的基本理论和应用情况。

1.1　遗传算法的产生与发展

早在20世纪50年代和60年代,就有少数几个计算机科学家独立地进行了所谓的"人工进化系统"研究,其出发点是进化的思想可以发展成为许多工程问题的优化工具。早期的研究形成了遗传算法的雏形,如大多数系统都遵行"适者生存"的仿自然法则,有些系统采用了基于种群(population)的设计方案,并且加入了自然选择和变异操作,还有一些系统对生物染色体编码进行了抽象处理,应用二进制编码。60年代初期,柏林工业大学的I. Rechenberg和H. P. Schwefel等在进行风洞实验时,由于设计中描述物体形状的参数难以用传统方法进行优化,因而利用生物变异的思想来随机改变参数值,并获得了较好的结果。随后,他们对这种方法进行了深入的研究,形成了进化计算的另一个分支——进化策略(Evolutionary Strategy, ES),如今ES和GA已呈融合之势。也是在20世纪60年代,L. J. Fogel等人在设计有限态自动机(Finite State Machine, FSM)时提出了进化规划(Evolutionary Programming, EP),他们借用进化的思想对一组FSM进行进化,以获得较好的FSM。他们将此方法应用到数据诊断、模式识别和分类及控制系统的设计等问题中,取得了较好的结果。后来又借助进化策略方法发展了进化规划,并用于数值优化及神经网络的训练等问题中。

由于缺乏一种通用的编码方案,人们只能依赖变异而非交叉来产生新的基因结构,早期的算法收效甚微。20世纪60年代中期,John Holland在A. S. Fraser和H. J. Bremermann等人工作的基础上提出了位串编码技术。这种编码既适用于变异操作,又适用于交叉(即杂交)操作,并且强调将交叉作为主要的遗传操作。随后,Holland将该算法用于自然和人工系统的自适应行为的研究中,并于1975年出版了其开创性著作"Adaptation in Natural and Artificial Sys-

tems”。以后，Holland 等人将该算法加以推广，应用到优化及机器学习等问题中，并正式定名为遗传算法。遗传算法的通用编码技术和简单有效的遗传操作为其广泛、成功地应用奠定了基础。Holland 早期有关遗传算法的许多概念一直沿用至今，可见 Holland 对遗传算法的贡献之大。他认为遗传算法本质上是适应算法，应用最多的是系统最优化的研究。

Holland 早期的工作集中在所谓的认知系统 CS 1(Cognitive System 1)的研究，借助最优化的方法获取学习的规则，遗传算法是他考虑的途径之一。于是他将基于遗传的机器学习(Genetic－based Machine Learning, GBML)方法发展成为 CS 1 的分类系统(Classifier System)学习方法，奠定了遗传算法重要思想的基础。遗传算法适用于最优化问题，归功于 Holland 的学生 De Jong，而 Grefenstette 开发了第一个遗传算法软件——称为 GENESIS，为遗传算法的普及推广起了重要作用。对遗传算法研究影响力最大的专著，要属于 1989 年美国伊利诺大学的 Goldberg 所著的“Genetic Algorithms in Search, Optimization, and Machine Learning”。这本书对于遗传算法理论及其多领域的应用展开了较为全面的分析和例证。1992 年，Michalewicz 出版了另一本很有影响力的著作“Genetic Algorithms + Data Structures = Evolution Programs”，对遗传算法应用于最优化问题起到了推波助澜的作用，1994 年该书又再版发行。

20 世纪 70 年代以来，关于遗传算法的博士论文，比较有代表性的有 A. D. Bethke 的“作为函数优化器的遗传算法”(密歇根大学，1980 年)、De Jong 的“一类遗传自适应系统的行为分析”(密歇根大学，1975 年)、T. E. Davis 的“从模拟退火收敛理论向简单遗传算法的外推”(佛罗里达大学，1991 年)。表 1.1 列出了遗传算法理论的经典研究成果。

表 1.1 遗传算法理论的经典研究成果

年份	贡献者	内容
1962	Holland	程序漫游元胞计算机自适应系统框架
1968	Holland	模式定理的建立
1971	Hollstein	具有交配和选择规则的二维函数优化
1972	Bosworth, Foo, Zeigler	提出具有复杂变异、类似于遗传算法的基因操作
1972	Frantz	位置非线性和倒位操作研究
1973	Holland	遗传算法中试验的最优配置和双臂强盗问题
1973	Martin	类似遗传算法的概率算法理论
1975	De Jong	用于 5 个测试函数的研究基本遗传算法基准参数
1975	Holland	出版了开创性著作《Adaptation in Natural and Artificial Systems》
1981	Bethke	应用 Walsh 函数分析模式
1981	Brindle	研究遗传算法中的选择和支配问题
1983	Pettit, Swigger	遗传算法应用于非稳定问题的粗略研究
1983	Wetzel	用遗传算法解决旅行商问题(TSP)
1984	Mauldin	基本遗传算法中用启发知识维持遗传多样性
1985	Baker	试验基于排序的选择方法
1985	Booker	建议采用部分匹配计分、分享操作和交配限制法

续表 1.1

年份	贡献者	内容
1985	Goldberg, Lingle	TSP 问题中采用部分匹配交叉
1985	Grefenstette, Fitzpattrick	对含噪声的函数进行测试
1985	Schaffer	多种群遗传算法解决多目标优化问题
1986	Goldberg	最优种群大小估计
1986	Grefenstette	元级遗传算法控制的遗传算法
1987	Baker	选择中随机误差的减少方法
1987	Goldberg	复制和交叉时最小欺骗问题(MDP)
1987	Goldberg, Richardson	借助分享函数的小生境和物种归纳法
1987	Goldberg, Segrest	复制和交叉的有限马尔可夫链
1987	Goldberg, Smith	双倍染色体遗传算法应用于非稳定函数优化
1987	Oliver, Smith, Holland	排列重组算子的模拟和分析
1987	Schaffer, Morishima	串编码自适应交叉试验
1987	Whitley	子孙测试应用于遗传算法的选择操作

20 余年来,遗传算法的应用无论是用来解决实际问题还是建模,其范围不断扩展,这主要依赖于遗传算法本身的逐渐成熟。近年来,许多冠以"遗传算法"的研究与 Holland 最初提出的算法已少有雷同之处,不同的遗传基因表达方式,不同的交叉和变异算子,特殊算子的引用,以及不同的再生和选择方法,但这些改进方法产生的灵感都来自大自然的生物进化,可以归为一个"算法簇"。人们用进化计算(Evolutionary Computation)来包容这样的遗传"算法簇"。它基本划分为四个分支:遗传算法(GA)、进化规划(EP)、进化策略(ES)和遗传程序设计(GP)。遗传算法研究热潮的兴起,人工智能再次成为人们关注的焦点。有些学者甚至提出,进化计算是人工智能的未来。其观点是,虽然我们不能设计人工智能(即用机器代替人的自然智能),但我们可以利用进化通过计算获得智能。目前,进化计算与人工神经网络、模糊系统理论一起已形成一个新的研究方向——计算智能(computational intelligence)。人工智能已从传统的基于符号处理的符号主义,向以神经网络为代表的连接主义和以进化计算为代表的进化主义方向发展。

应该说,20 世纪 80 年代中期以来是遗传算法和进化计算的蓬勃发展期。以遗传算法、进化计算为主题的多个国际会议在世界各地定期召开。1985 年,在美国卡耐基·梅隆大学召开的第一届国际遗传算法会议 ICGA'85,以后该会议每隔一年举行一次。1997 年夏季在美国密歇根大学召开了 ICGA'97 。现在与之平行进行的国际会议很多,其中 International Conference on Evolutionary Programming 和 IEEE International Conference on Evolutionary Computation 也分别召开了 6 届和 4 届。此外,每年夏季在美国斯坦福大学召开有关遗传程序设计的国际会议(The Annual Conference of Genetic Programming)。有关遗传算法基础理论的学术活动也很活跃,第一届遗传算法与分类系统研讨会(The First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, FOGA/CS)1990 年在美国印第安那大学召开,以后

每隔两年召开一次，从会议论文中选编的论文集："Foundations of Genetic Algorithms Ⅰ"和"Foundations of Genetic Algorithms Ⅱ"两卷已由MIT出版社出版发行。在欧洲"Parallel Problem Solving from Nature：FPSN"为题的国际会议从1990年开始在德国举行以来，在比利时和德国隔年轮流举行。同样值得一提的是，国际互联网上也有多种相关的 mailing list，Usenet上还有专门的新闻组 comp.ai.genetic。由于进化计算应用广泛，一些杂志及国际会议论文集中都有这方面的文章，现在还出版了两种关于进化计算的新杂志"Evolutionary Computation"和"IEEE Transactions on Evolutionary Computation"，一些国际性期刊也竞相出版这方面的专刊。另外，日本新的计算机发展规划RWC计划(Real World Computing Program)也把遗传算法、进化计算作为其主要支撑技术之一，用来进行信息的集成、学习及组织等。

1980年以来，人们越来越清楚地意识到传统人工智能方法的局限性，而且随着计算机速度的提高及并行计算机的普及，遗传算法和进化计算对机器速度的要求已不再是制约其发展的因素。德国Dortmund大学1993年末的一份研究报告表明，根据不完全统计，进化算法已在16个大领域、250多个小领域中获得了应用。遗传算法在机器学习、过程控制、经济预测、工程优化等领域取得的成功，已引起了数学、物理学、化学、生物学、计算机科学、社会科学、经济学及工程应用等领域专家的极大兴趣。某些学者研究了进化计算的突现行为(emergent behavior)后声称，进化计算与混沌理论、分形几何将成为人们研究非线性现象和复杂系统的新的三大方法，并将与神经网络一起成为人们研究认知过程的重要工具。20世纪90年代以后，人们比较重视遗传算法的一些基本问题，De Jong称为"重访基本的假设"，这方面的研究内容主要有①表示和形态发生学；②拉马克算子等的引入；③非随机配对和物种形成；④分散的、高度并行的模型；⑤自适应系统；⑥共同进化系统。同时由于遗传算法在应用研究方面的长处主要得益于其求解的有效性、现有仿真环境下易于实现、可扩充性和易于与其他方法相结合，可以预料在不远的将来，随着理论研究的不断深入和应用领域的不断拓广，遗传算法和进化计算将取得长足的发展。

我国有关遗传算法、进化计算的研究，从20世纪90年代以来一直处于不断上升的时期，特别是近年来，遗传算法、进化计算的应用在许多领域取得了令人瞩目的成果。据不完全统计，1997～1999年三年间发表在国内二级以上学术刊物上有关遗传算法、进化计算的文章接近200篇左右，该类研究获得不同渠道的经费资助比例也在逐年上升。武汉大学刘勇、康立山等于1995年出版了《非数值并行计算(第2册)——遗传算法》；陈国良、王煦法等于1996年出版《遗传算法及其应用》；潘正军、康立山等于1998年出版了《演化计算》；周明、孙树栋于1999年出版了《遗传算法原理及其应用》。国内有关遗传算法的BBS电子公告牌有国家智能中心曙光站 bbs.neic.ac.cn、北京大学阳光创意站 bbs.pku.edu.cn、清华大学水木清华站 bbs.net.tsinghua.edu.cn、西安交通大学兵马俑站 bbs.xanet.edu.cn 等。

本书附录Ⅰ介绍了有关遗传算法及进化计算的国内、外学术组织及其活动情况。

1.2 遗传算法概要

1.2.1 生物进化理论和遗传学的基本知识

在介绍遗传算法之前，有必要了解有关的生物进化理论和遗传学的基本知识。

我们知道，生命的基本特征包括生长、繁殖、新陈代谢和遗传与变异。生命是进化的产物，

现代的生物是在长期进化过程中发展起来的。达尔文(1858年)用自然选择(natural selection)来解释物种的起源和生物的进化,其自然选择学说包括以下三个方面:

(1) **遗传(heredity)** 这是生物的普遍特征,"种瓜得瓜,种豆得豆",亲代把生物信息交给子代,子代按照所得信息而发育、分化,因而子代总是和亲代具有相同或相似的性状。生物有了这个特征,物种才能稳定存在。

(2) **变异(variation)** 亲代和子代之间以及子代的不同个体之间总有些差异,这种现象,称为变异。变异是随机发生的,变异的选择和积累是生命多样性的根源。

(3) **生存斗争和适者生存** 自然选择来自繁殖过剩和生存斗争。由于弱肉强食的生存斗争不断地进行,其结果是适者生存,具有适应性变异的个体被保留下来,不具有适应性变异的个体被淘汰,通过一代代的生存环境的选择作用,物种变异被定向着一个方向积累,于是性状逐渐和原先的祖先种不同,演变为新的物种。这种自然选择过程是一个长期的、缓慢的、连续的过程。

达尔文的进化理论是生物学史上的一个重要里程碑,它解释了自然选择作用下生物的渐变式进化。1866年孟德尔发表了"植物杂交实验"的论文,他提出的遗传学的两个基本规律——分离律和自由组合律,奠定了现代遗传学的基础。随着细胞学的发展,染色体、减数分裂和受精过程相继被发现,Water S. Sutton发现染色体的行为与基因的遗传因子行为是平行的,因此提出遗传因子是位于染色体上的。美国遗传学家摩尔根(T. H. Morgan)进一步确立了染色体的遗传学说,认为遗传形状是由基因决定的,染色体的变化必然在遗传形状上有所反映。生物的形状往往不是简单地决定于单个基因,而是不同基因相互作用的结果,基因表达要求一定的环境条件,同一基因型在不同的环境条件下可以产生不同的表现型。20世纪20年代以来,随着遗传学的发展,一些科学家用统计生物学和种群遗传学的成就重新解释达尔文的自然选择理论,他们通过精确地研究种群基因频率由一代到下一代的变化,来阐述自然选择是如何起作用的,形成现代综合进化论(synthetic theory of evolution)。种群遗传学是以种群为单位而不是以个体为单位的遗传学,是研究种群中基因的组成及其变化的生物学。在一定地域中,一个物种的全体成员构成一个种群(population),种群的主要特征是种群内的雌雄个体能够通过有性生殖实现基因的交流。生物的进化实际上是种群的进化,个体总是要消亡,但种群则是继续保留,每一代个体基因型的改变会影响种群基因库(gene pool)的组成。而种群基因库组成的变化就是这一种群的进化,没有所谓的生存斗争问题,单是个体繁殖机会的差异也能造成后代遗传组成的改变,自然选择也能够进行。综合进化论对达尔文式的进化给予了新的更加精确的解释。

生物进化非常复杂,现有的进化理论所不能解释的问题比已经解释的问题还要多。除了达尔文的渐变进化外,人们又提出了很多新的非达尔文式进化理论,如木村资生的分子进化中性理论(neutral theory of molecular evolution)、Goldschmidt的跳跃进化(saltation)、N. Eldredge的间断平衡进化(Punctuated Equilibrium Evolution)等。随着生物学的前沿领域——生物物理、分子生物学和生物化学的发展,关于生物进化的理论仍在发展之中,但以自然选择为核心的进化理论比其他学说的影响广泛而深远,它仍然是各种生物进化理论的一个重要基础。

遗传算法模拟的是怎样的生物进化模型呢?假设对相当于自然界中的一群人的一个种群进行操作,第一步的选择是以现实世界中的优胜劣汰现象为背景的;第二步的重组交叉则相当于人类的结婚和生育;第三步的变异则与自然界中偶然发生的变异是一致的。人类偶然出现

的返祖现象便是一种变异。由于包含着对模式的操作,遗传算法不断地产生出更加优良的个体,正如人类向前进化一样。所采用的遗传操作都与生物尤其是人类的进化过程相对应。如果我们再仔细分析遗传算法的操作对象种群,实际上它对应的是一群人,而不是整个人类。一群人随着时间的推移而不断地进化,并具备越来越多的优良品质。然而,由于他们的生长、演变、环境和原始祖先的局限性,经过相当一段时间后,他们将逐渐进化到某些特征相对优势的状态(例如中国人都是黄皮肤、黑眼睛以及特有的文化和社会传统习惯),我们定义这种状态为平衡态。当一个种群进化到这种状态,这种种群的特性就不再有很大的变化了。一个简单的遗传算法,从初始代开始,并且各项参数都设定,也会达到平衡态。此时种群中的优良个体仅包含了某些类的优良模式,因为该遗传算法的设置特性参数使得这些优良模式的各个串位未能得到平等的竞争机会。

现实世界中有许多民族,每个民族都有各自的优缺点。历史上民族之间通过多种形式的交流(包括战争、移民等),打破了各个民族的平衡态,从而推动他们达到更高层次的平衡态,使整个人类向前进化。现实生活中的例子可以在生物实验室中找到,为了改良动、植物品种,常常采用杂交、嫁接等措施,即是为了这个目的。

既然遗传算法效法基于自然选择的生物进化,是一种模仿生物进化过程的随机方法。下面先给出几个生物学的基本概念与术语,这对于理解遗传算法是非常重要的。

染色体(chromosome) 生物细胞中含有的一种微小的丝状化合物。它是遗传物质的主要载体,由多个遗传因子——基因组成。

脱氧核糖核酸(DNA) 控制并决定生物遗传性状的染色体主要是由一种叫做脱氧核糖核酸(deoxyribonucleic acid 简称 DNA)的物质构成。DNA 在染色体中有规则地排列着,它是个大分子的有机聚合物,其基本结构单位是核苷酸。每个核苷酸有四种称为碱基的环状有机化合物中的一种、一分子戊糖和磷酸分子组成。许多核苷酸通过磷酸二酯键相结合形成一条长长的链状结构,两个链状结构再通过碱基间的氢键有规律地扭合在一起,相互卷曲起来形成一种双螺旋结构。

核糖核酸(RNA) 低等生物中含有一种核糖核酸(ribonucleic acid,简称 RNA)的物质,它的作用和结构与 DNA 类似。

遗传因子(gene) DNA 或 RNA 长链结构中占有一定位置的基本遗传单位,也称为基因。生物的基因数量根据物种的不同多少不一,小的病毒只含有几个基因,而高等动、植物的基因却数以万计。一个基因或多个基因决定了组成蛋白质的 20 种氨基酸的组成比例及其排列顺序。

遗传子型(genotype) 遗传因子组合的模型叫遗传子型。它是性状染色体的内部表现,又称基因型。一个细胞核中所有染色体所携带的遗传信息的全体称为一个基因组(genome)。

表现型(phenotype) 由染色体决定性状的外部表现,或者说,根据遗传子型形成的个体,称为表现型。

基因座(locus) 遗传基因在染色体中所占据的位置。同一基因座可能有的全部基因称为等位基因(allele)。

个体(individual) 指染色体带有特征的实体。

种群(population) 染色体带有特征的个体的集合称为种群。该集合内个体数称为群体的大小。有时个体的集合也称为个体群。

进化(evolution) 生物在其延续生存的过程中,逐渐适应其生存环境,使得其品质不断得到改良,这种生命现象称为进化。生物的进化是以种群的形式进行的。

适应度(fitness) 在研究自然界中生物的遗传和进化现象时,生物学家使用适应度这个术语来度量某个物种对于生存环境的适应程度。对生存环境适应程度较高的物种将获得更多的繁殖机会,而对生存环境适应程度较低的物种,其繁殖机会就会相对较少,甚至逐渐灭绝。

选择(selection) 指决定以一定的概率从种群中选择若干个体的操作。一般而言,选择的过程是一种基于适应度的优胜劣汰的过程。

复制(reproduction) 细胞在分裂时,遗传物质DNA通过复制而转移到新产生的细胞中,新的细胞就继承了旧细胞的基因。

交叉(crossover) 有性生殖生物在繁殖下一代时两个同源染色体之间通过交叉而重组,亦即在两个染色体的某一相同位置处DNA被切断,其前后两串分别交叉组合形成两个新的染色体。这个过程又称基因重组recombination,俗称"杂交"。

变异(mutation) 在细胞进行复制时可能以很小的概率产生某些复制差错,从而使DNA发生某种变异,产生出新的染色体,这些新的染色体表现出新的性状。

编码(coding) DNA中遗传信息在一个长链上按一定的模式排列,也即进行了遗传编码。遗传编码可以看作从表现型到遗传子型的映射。

解码(decoding) 从遗传子型到表现型的映射。

1.2.2 遗传算法的基本思想

现在,我们引用上面的术语来更好地描述遗传算法的基本思想。遗传算法是从代表问题可能潜在解集的一个种群(population)开始的,而一个种群则由经过基因(gene)编码(coding)的一定数目的个体(individual)组成。每个个体实际上是染色体(chromosome)带有特征的实体。染色体作为遗传物质的主要载体,即多个基因的集合,其内部表现(即基因型)是某种基因组合,它决定了个体的形状的外部表现,如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此,在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂,我们往往进行简化,如二进制编码。初代种群产生之后,按照适者生存和优胜劣汰的原理,逐代(generation)演化产生出越来越好的近似解。在每一代,根据问题域中个体的适应度(fitness)大小挑选(selection)个体,并借助于自然遗传学的遗传算子(genetic operators)进行组合交叉(crossover)和变异(mutation),产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境,末代种群中的最优个体经过解码(decoding),可以作为问题近似最优解。

遗传算法采纳了自然进化模型,如选择、交叉、变异、迁移、局域与邻域等。图1.1表示了基本遗传算法的过程。计算开始时,一定数目N个个体(父个体1、父个体2、父个体3、父个体4……)即种群随机地初始化,并计算每个个体的适应度函数,第一代也即初始代就产生了。如果不满足优化准则,开始产生新一代的计算。为了产生下一代,按照适应度选择个体,父代要求基因重组(交叉)而产生子代。所有的子代按一定概率变异。然后子代的适应度又被重新计算,子代被插入到种群中将父代取而代之,构成新的一代(子个体1、子个体2、子个体3、子个体4……)。这一过程循环执行,直到满足优化准则为止。

尽管这样单一种群的遗传算法很强大,可以很好地解决相当广泛的问题。但采用多种群即有子种群的算法往往会获得更好的结果。每个子种群像单种群遗传算法一样独立地演算若

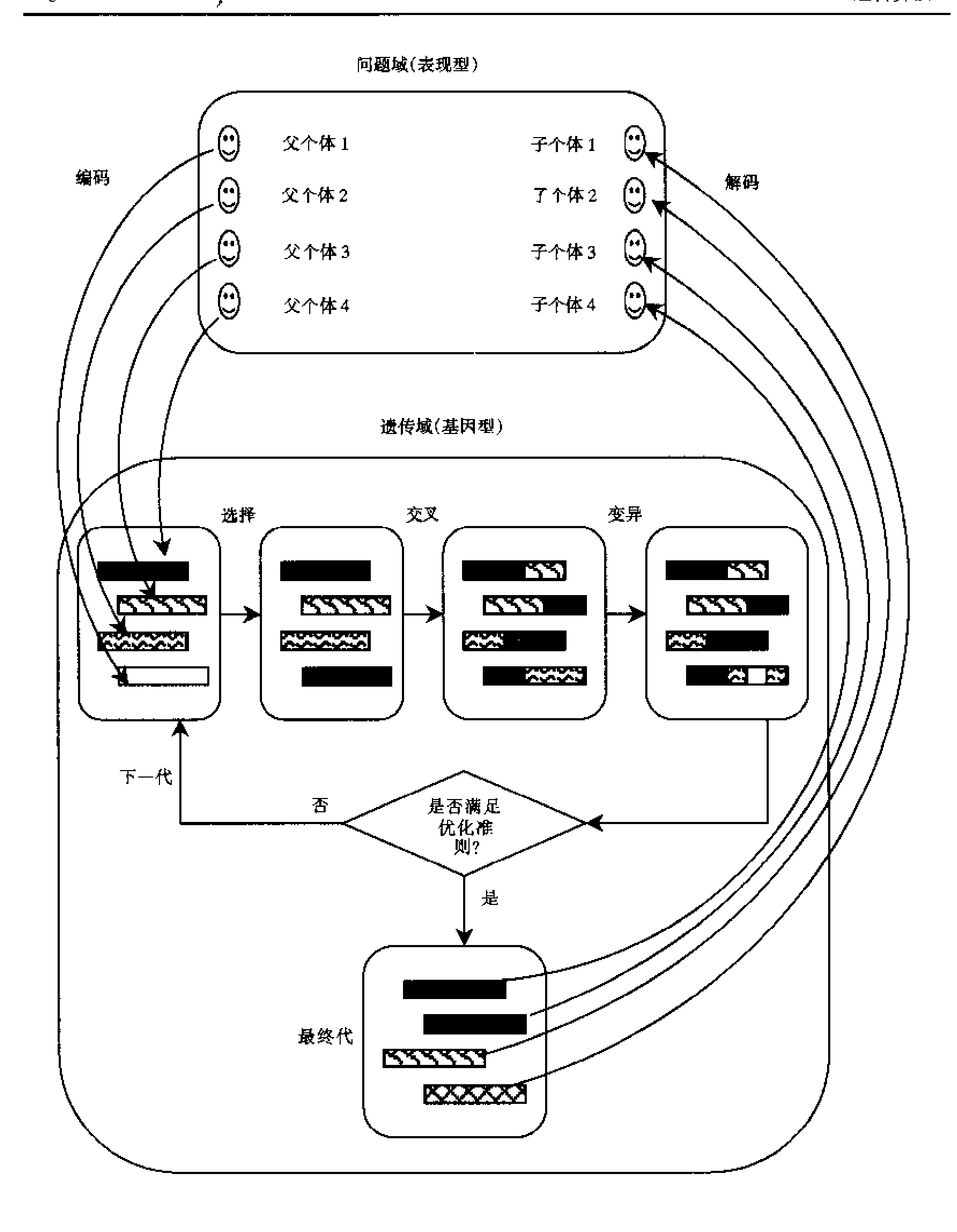


图 1.1　遗传算法的过程

干代后，在子种群之间进行个体交换。这种多种群遗传算法更加贴近于自然中种族的进化，称为并行遗传算法(Paralleling Genetic Algorithm, PGA)，此算法在第 4 章中将详细介绍。

1.2.3 遗传算法的特点

我们知道,传统的优化方法主要有三种:枚举法、启发式算法和搜索算法:

(1) *枚举法* 枚举出可行解集合内的所有可行解,以求出精确最优解。对于连续函数,该方法要求先对其进行离散化处理,这样就可能因离散处理而永远达不到最优解。此外,当枚举空间比较大时,该方法的求解效率比较低,有时甚至在目前先进计算工具上无法求解。

(2) *启发式算法* 寻求一种能产生可行解的启发式规则,以找到一个最优解或近似最优解。该方法的求解效率比较高,但对每一个需求解的问题必须找出其特有的启发式规则,这个启发式规则一般无通用性,不适合于其他问题。

(3) *搜索算法* 寻求一种搜索算法,该算法在可行解集合的一个子集内进行搜索操作,以找到问题的最优解或者近似最优解。该方法虽然保证不了一定能够得到问题的最优解,但若适当地利用一些启发知识,就可在近似解的质量和效率上达到一种较好的平衡。

随着问题种类的不同以及问题规模的扩大,要寻求一种能以有限的代价来解决搜索和优化的通用方法,遗传算法正是为我们提供的一个有效的途径,它不同于传统的搜索和优化方法。主要区别在于:

① 自组织、自适应和自学习性(智能性)。应用遗传算法求解问题时,在编码方案、适应度函数及遗传算子确定后,算法将利用进化过程中获得的信息自行组织搜索。由于基于自然的选择策略为"适者生存,不适应者被淘汰",因而适应度大的个体具有较高的生存概率。通常,适应度大的个体具有更适应环境的基因结构,再通过基因重组和基因突变等遗传操作,就可能产生更适应环境的后代。进化算法的这种自组织、自适应特征,使它同时具有能根据环境变化来自动发现环境的特性和规律的能力。自然选择消除了算法设计过程中的一个最大障碍,即需要事先描述问题的全部特点,并要说明针对问题的不同特点算法应采取的措施。因此,利用遗传算法的方法,我们可以解决那些复杂的非结构化问题。

② 遗传算法的本质并行性。遗传算法按并行方式搜索一个种群数目的点,而不是单点。它的并行性表现在两个方面,一是遗传算法是内在并行的(inherent parallelism),即遗传算法本身非常适合大规模并行。最简单的并行方式是让几百甚至数千台计算机各自进行独立种群的演化计算,运行过程中甚至不进行任何通信(独立的种群之间若有少量的通信一般会带来更好的结果),等到运算结束时才通信比较,选取最佳个体。这种并行处理方式对并行系统结构没有什么限制和要求,可以说,遗传算法适合在目前所有的并行机或分布式系统上进行并行处理,而且对并行效率没有太大影响。二是遗传算法的内含并行性(implicit parallelism)。由于遗传算法采用种群的方式组织搜索,因而可同时搜索解空间内的多个区域,并相互交流信息。使用这种搜索方式,虽然每次只执行与种群规模 n 成比例的计算,但实质上已进行了大约 $O(n^3)$ 次有效搜索,这就使遗传算法能以较少的计算获得较大的收益。

③ 遗传算法不需要求导或其他辅助知识,而只需要影响搜索方向的目标函数和相应的适应度函数。

④ 遗传算法强调概率转换规则,而不是确定的转换规则。

⑤ 遗传算法可以更加直接地应用。

⑥ 遗传算法对给定问题,可以产生许多的潜在解,最终选择可以由使用者确定(在某些特殊情况下,如多目标优化问题不止一个解存在,有一组 pareto 最优解。这种遗传算法对于确认可替代解集而言是特别合适的)。

1.3 遗传算法的基本操作

遗传算法包括三个基本操作:选择、交叉和变异。这些基本操作又有许多不同的方法,下面我们逐一进行介绍。

1. 选择(selection)

选择是用来确定重组或交叉个体,以及被选个体将产生多少个子代个体。首先计算适应度:

① 按比例的适应度计算(proportional fitness assignment);

② 基于排序的适应度计算(rank - based fitness assignment)。

适应度计算之后是实际的选择,按照适应度进行父代个体的选择。可以挑选以下的算法:

① 轮盘赌选择(roulette wheel selection);

② 随机遍历抽样(stochastic universal sampling);

③ 局部选择(local selection);

④ 截断选择(truncation selection);

⑤ 锦标赛选择(tournament selection)。

2. 交叉或基因重组(crossover / recombination)

基因重组是结合来自父代交配种群中的信息产生新的个体。依据个体编码表示方法的不同,可以有以下的算法:

① 实值重组(real valued recombination)

- 离散重组(discrete recombination);
- 中间重组(intermediate recombination);
- 线性重组(linear recombination);
- 扩展线性重组(extended linear recombination)。

② 二进制交叉(binary valued crossover)

- 单点交叉(single - point crossover);
- 多点交叉(multiple - point crossover);
- 均匀交叉(uniform crossover);
- 洗牌交叉(shuffle crossover);
- 缩小代理交叉(crossover with reduced surrogate)。

3. 变异(mutation)

交叉之后子代经历的变异,实际上是子代基因按小概率扰动产生的变化。依据个体编码表示方法的不同,可以有以下的算法:

① 实值变异;

② 二进制变异。

至此,可能读者对以上的不同算法还很陌生,我们将它们留在第 2 章详细解释。这里我们结合一个简单的实例考察一下二进制编码的轮盘赌选择、单点交叉和变异操作。

图 1.2 所示的是一组二进制基因码构成的个体组成的初始种群,个体的适应度评价值经计算由括号内的数值表示,适应度越大代表这个个体越好。

0001100000 (8)	0101111001 (5)	0000000101 (2)	1001110100 (10)	1010101010 (7)
1110010110 (12)	1001011011 (5)	1100000001 (19)	1001110100 (10)	0001010011 (14)

图 1.2 初始种群的分布

轮盘赌选择方法类似于博彩游戏中的轮盘赌。如图 1.3 所示，个体适应度按比例转化为选中概率，将轮盘分成 10 个扇区，因为要进行 10 次选择，所以产生 10 个[0,1]之间的随机数，相当于转动 10 次轮盘，获得 10 次转盘停止时指针位置，指针停止在某一扇区，该扇区代表的个体即被选中。

个体	染色体	适应度	选择概率	累积概率
1	0001100000	8	0.086 957	0.086 957
2	0101111001	5	0.054 348	0.141 304
3	0000000101	2	0.021 739	0.163 043
4	1001110100	10	0.108 696	0.271 739
5	1010101010	7	0.076 087	0.347 826
6	1110010110	12	0.130 435	0.478 261
7	1001011011	5	0.054 348	0.532 609
8	1100000001	19	0.206 522	0.739 130
9	1001110100	10	0.108 696	0.847 826
10	0001010011	14	0.152 174	1.000 000

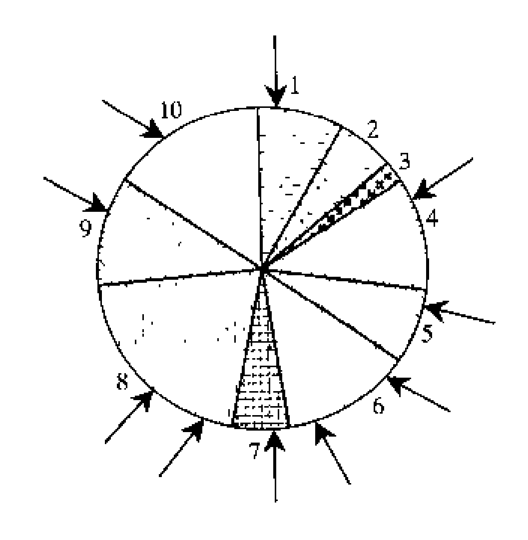


图 1.3 轮盘赌选择

假设产生随机数序列为 0.070 221，0.545 929，0.784 567，0.446 93，0.507 893，0.291 198，0.716 34，0.272 901，0.371 435，0.854 641，将该随机序列与计算获得的累积概率比较，则依次序号为 1，8，9，6，7，5，8，4，6，10 个体被选中。显然适应度高的个体被选中的概率

大,而且可能被选中;而适应度低的个体则很有可能被淘汰。在第一次生存竞争考验中,序号为 2 的个体(0101111001)和 3 的个体(0000000101)被淘汰,代之以适应度较高的个体 8 和 6,这个过程被称为再生(reproduction)。再生之后重要的遗传操作是交叉,在生物学上称为杂交,可以视为生物之所以得以进化之所在。我们以单点交叉(one - point crossover)为例,任意挑选经过选择操作后种群中两个个体作为交叉对象,即两个父个体经过染色体交换重组产生两个子个体,如图 1.4 所示。随机产生一个交叉点位置,父个体 1 和父个体 2 在交叉点位置之右的部分基因码互换,形成子个体 1 和子个体 2。类似地完成其他个体的交叉操作。

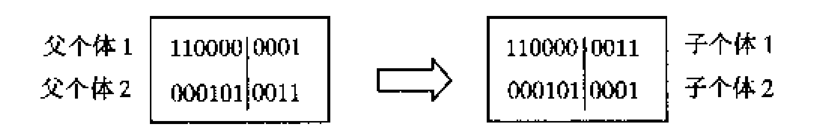


图 1.4 单点交叉

如果只考虑交叉操作实现进化机制,在多数情况下是不行的,这与生物界近亲繁殖影响进化历程是类似的。因为,种群的个体数是有限的,经过若干代交叉操作,因为源于一个较好祖先的子个体逐渐充斥整个种群的现象,问题会过早收敛(premature convergence),当然,最后获得的个体不能代表问题的最优解。为避免过早收敛,有必要在进化过程中加入具有新遗传基因的个体。解决办法之一是效法自然界生物变异。生物性状的变异实际上是控制该性状的基因码发生了突变,这对于保持生物多样性是非常重要的。模仿生物变异的遗传操作,对于二进制的基因码组成的个体种群,实现基因码的小概率翻转,即达到变异的目的。

如图 1.5 所示,对于个体 1001110100 产生变异,以小概率决定第 4 个遗传因子翻转,即将 1 换为 0。

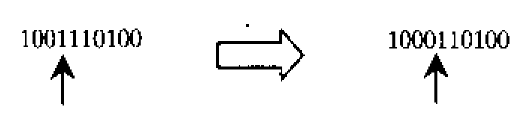


图 1.5 变异

一般而言,一个世代的简单进化过程就包括了基于适应度的选择和再生、交叉和变异操作。将上面的所有种群的遗传操作综合起来,初始种群的第一代进化过程如图 1.6 所示。初始种群经过选择操作,适应度较高的 8 号和 6 号个体分别复制出 2 个,适应度较低的 2 号和 3 号遭到淘汰,接下来按一定概率选择了 4 对父个体分别完成交叉操作,在随机确定的“|”位置实行单点交叉生成 4 对子个体。最后按小概率选中某个个体的基因码位置,产生变异。这样经过上述过程便形成了第一代的群体。以后一代一代的进化过程如此循环下去,每一代结束都产生新的种群。演化的代数主要取决于代表问题解的收敛状态,末代种群中最佳个体作为问题的最优近似解。

遗传算法进化模式如图 1.7 所示,搜索空间中个体演变为最优个体,其在高适应度上的增殖概率是按世代递增的,图中表现个体的色彩浓淡表示个体增殖的概率分布。

遗传算法的一般流程如图 1.8 所示:

第 1 步　随机产生初始种群,个体数目一定,每个个体表示为染色体的基因编码;

第 2 步　计算个体的适应度,并判断是否符合优化准则,若符合,输出最佳个体及其代表的最优解,并结束计算;否则转向第 3 步;

第 3 步　依据适应度选择再生个体,适应度高的个体被选中的概率高,适应度低的个体可

初始种群

0001100000 (8)	0101111001 (5)	0000000101 (2)	1001110100 (10)	1010101010 (7)
1110010110 (12)	1001011011 (5)	1100000001 (19)	1001110100 (10)	0001010011 (14)

↓ 选择、再生

0001|100000	111|0010110	110000|0001	1001110100	1010101|010
1110|010110	100|1011011	100111|0100	1100000001	0001010|011

↓ 交叉

0001010110	1111011011	1100000100	1001110100	1010101011
1110100000	1000010110	1001110001	1100000001	0001010010

↓ 变异

0001010110	1111011011	1100000100	1001110100	1010101011
1110100000	1000010110	1000110001	1100000001	0001010010

图 1.6　遗传算法的进化过程

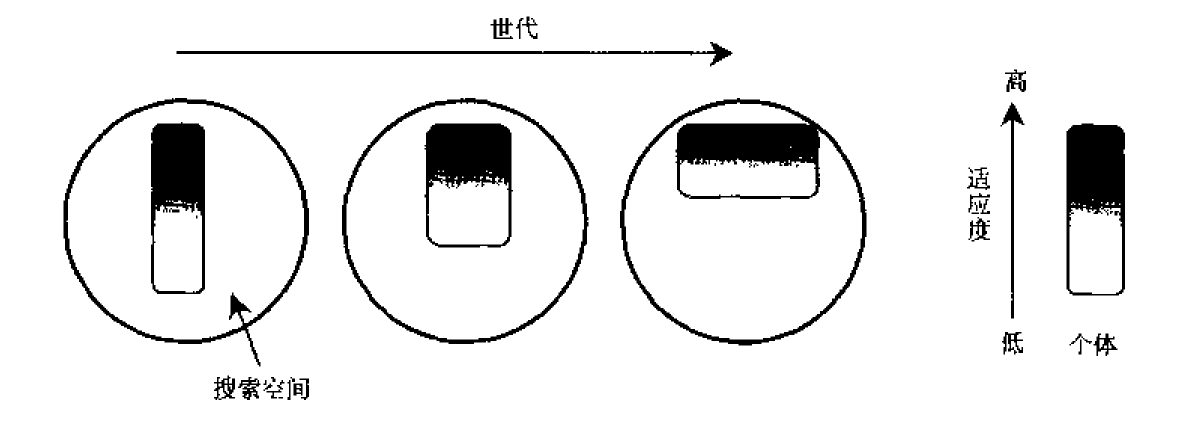


图 1.7　遗传算法进化模式示意图

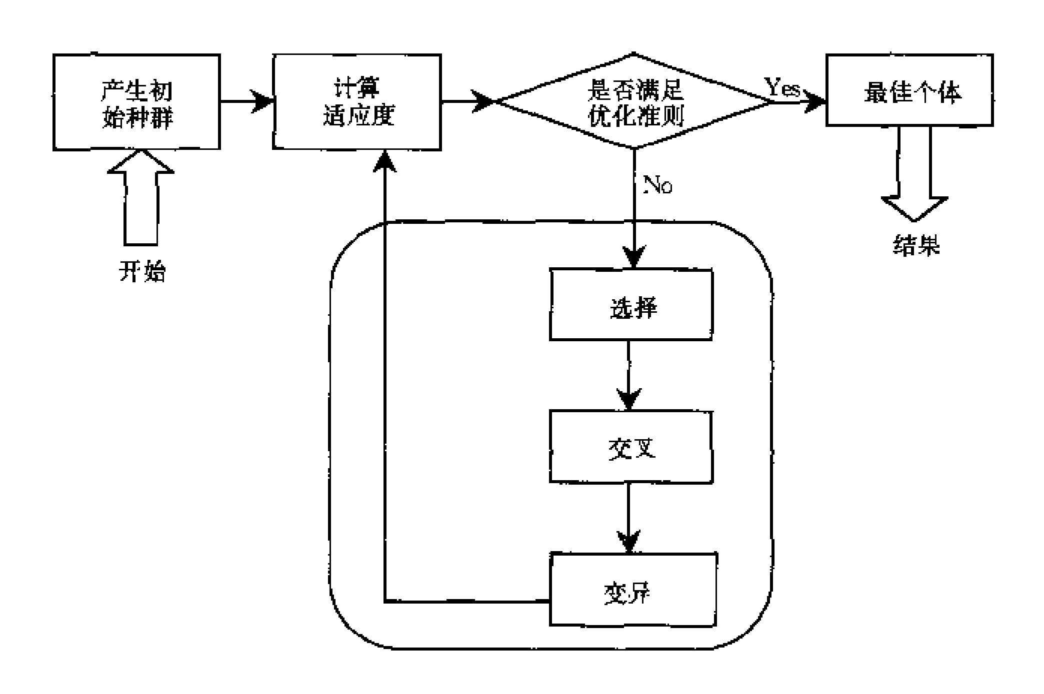


图 1.8　遗传算法的流程图

能被淘汰；

第 4 步　按照一定的交叉概率和交叉方法，生成新的个体；

第 5 步　按照一定的变异概率和变异方法，生成新的个体；

第 6 步　由交叉和变异产生新一代的种群，返回到第 2 步。

遗传算法中的优化准则，一般依据问题的不同有不同的确定方式。例如，可以采用以下的准则之一作为判断条件：

① 种群中个体的最大适应度超过预先设定值；

② 种群中个体的平均适应度超过预先设定值；

③ 世代数超过预先设定值。

1.4　遗传算法的应用情况

遗传算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的鲁棒性，所以广泛应用于很多学科。下面是遗传算法的一些主要应用领域：

(1) **函数优化**　函数优化是遗传算法的经典应用领域，也是对遗传算法进行性能评价的常用算例。很多人构造出了各种各样的复杂形式的测试函数，有连续函数也有离散函数，有凸函数也有凹函数，有低维函数也有高维函数，有确定函数也有随机函数，有单峰函数也有多峰函数等，人们用这些几何特性各异的函数来评价遗传算法的性能。而对于一些非线性、多模型、多目标的函数优化问题，用其他优化方法较难求解，遗传算法却可以方便地得到较好的结果。

(2) **组合优化**　随着问题规模的扩大，组合优化问题的搜索空间急剧扩大，有时在目前的计算机上用枚举法很难或者甚至不可能得到其精确最优解。对于这类复杂问题，人们已意识

到应把精力放在寻求其满意解上,而遗传算法则是寻求这种满意解的最佳工具之一。实践证明,遗传算法对于组合优化中的NP完全问题非常有效。例如,遗传算法已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面得到成功的应用。

(3) **生产调度问题** 生产调度问题在许多情况下所建立起来的数学模型难以精确求解,即使经过一些简化之后可以进行求解,也会因简化太多而使得求解结果与实际相差甚远。因此,目前在现实生产中也主要靠一些经验进行调度。遗传算法已成为解决复杂调度问题的有效工具,在单件生产车间调度、流水线生产车间调度、生产规划、任务分配等方面遗传算法都得到了有效的应用。

(4) **自动控制** 在自动控制领域中许多与优化相关的问题需要求解,遗传算法的应用日益增加,并显示了良好的效果。例如用遗传算法进行航空控制系统的优化、基于遗传算法的模糊控制器优化设计、基于遗传算法的参数辨识、利用遗传算法进行人工神经网络的结构优化设计和权值学习,都显示出了遗传算法在这些领域中应用的可能性。

(5) **机器人智能控制** 机器人是一类复杂的难以精确建模的人工系统,而遗传算法的起源就来自于对人工自适应系统的研究,所以机器人智能控制理所当然地成为遗传算法的一个重要应用领域。例如遗传算法已经在移动机器人路径规划、关节机器人运动轨迹规划、机器人逆运动学求解、细胞机器人的结构优化和行动协调等方面得到研究和应用。

(6) **图像处理和模式识别** 图像处理和模式识别是计算机视觉中的一个重要研究领域。在图像处理过程中,如扫描、特征提取、图像分割等不可避免地会产生一些误差,这些误差会影响到图像处理和识别的效果。如何使这些误差最小是使计算机视觉达到实用化的重要要求。遗传算法在图像处理中的优化计算方面是完全胜任的。目前已在图像恢复、图像边缘特征提取、几何形状识别等方面得到了应用。

(7) **人工生命** 人工生命是用计算机等人工媒体模拟或构造出具有自然生物系统特有行为的人造系统。自组织能力和自学习能力是人工生命的两大主要特征。人工生命与遗传算法有着密切的关系,基于遗传算法的进化模型是研究人工生命现象的重要理论基础。虽然人工生命的研究尚处于启蒙阶段,但遗传算法已在其进化模型、学习模型、行为模型等方面显示了初步的应用能力。可以预见,遗传算法在人工生命及复杂自适应系统的模拟与设计、复杂系统突现性理论研究中,将得到更为深入的发展。

(8) **遗传程序设计** Koza发展了遗传程序设计的概念,他使用了以LISP语言所表示的编码方法,基于对一种树型结构所进行的遗传操作自动生成计算机程序。虽然遗传程序设计的理论尚未成熟,应用也有一些限制,但它已有一些成功的应用。

(9) **机器学习** 学习能力是高级自适应系统所应具备的能力之一。基于遗传算法的机器学习,特别是分类器系统,在许多领域得到了应用。例如,遗传算法被用于模糊控制规则的学习,利用遗传算法学习隶属度函数,从而更好地改进了模糊系统的性能。基于遗传算法的机器学习可用于调整人工神经网络的连接权,也可用于神经网络结构的优化设计。分类器系统在多机器人路径规划系统中得到了成功的应用。

参考文献

[1] Holland J H. Adaptation in Natural and Artificial Systems. MIT Press, 1975

[2] Goldberg D E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA, Addison – Wisely, 1989

[3] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer – Verlag, Second, Extended Edition, 1994

[4] Hofbauer J, Sigmund K. The Theory of Evolution and Dynamical Systems. Cambridge University Press, 1988

[5] Collins R J. Studies in Artificial Evolution. Doctoral Dissertation, Artificial Life Laboratory, Department of Computer Science, University of California, 1992

[6] Angeline P J. Evolutionary Algorithms and Emergent Intelligence. Doctoral Dissertation, The Ohio State University, 1993

[7] Muhlenbein H. Evolutionary Algorithms: Theory and Applications. GMD Schloss Birlinhoven, 1995

[8] Grefenstette J J. GENESIS: A System for Using Genetic Search Procedures. In: Proceedings of the 1984 Conference on Intelligent Systems and Machines, 1984, 161～165

[9] Grefenstette J J(ed). Proceedings of the First International Conference on the Genetic Algorithms and their Applications. Lawrence Erlbaum Associates, Publishers, 1985

[10] Forrest S(ed). Genetic Algorithms, Proceedings of the Fifth International Conference on Genetic Algorithms. Morgan Kaufmann Publishers, 1993

[11] Winter G(ed). Genetic Algorithms in Engineering and Computer Science. Wiley, 1995

[12] Whitley L.D(ed). Foundations of Genetic Algorithms II. Morgan Kaufmann Publishers, 1993

[13] Forrest S, Mitchell M. What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation. In: Proceedings of the Fourth International Conference on Genetic Algorithms, CA, Morgan Kaufman, 1991, 120～131

[14] Bonissone P, Goebel K. Hybrid Soft Computing System: Industrial and Commercial Applications. In: Proceedings of the IEEE, 1999, 87(9): 1641～1665

[15] Nilsson N J. Artificial Intelligence: A New Synthesis. Morgan Kaufman & 机械工业出版社, 1999

[16] 安居院猛，長尾智晴. ジエネティックアルゴリズム. 昭晃堂，1993

[17] 北野宏明. 遺伝的アルゴリズム. 産業図書，1993

[18] 田中一男. インテリジェント制御システム. 共立出版株式会社，1994

[19] Nirwan A, Edwin H 著. 用于最优化的计算智能. 李军，边肇祺 译. 北京：清华大学出版社，1999

[20] 凯文·凝维克 著. 机器的征途. 李碧 等译. 呼和浩特：内蒙古人民出版社，1998

[21] 理查德·道金斯 著. 自私的基因. 卢允中 等译. 长春：吉林人民出版社，1998

[22] 特瑞·波索马特尔，大卫·格林 著. 沙地上的图案——计算机、复杂和生命. 陈禹 等译. 南昌：江西教育出版社，1999

[23] 迈克儿·J·贝希 著. 达尔文的黑匣子——生化理论对进化论的挑战. 刑锡范 等译. 中央编译出版社，1998

[24] 米歇尔·沃尔德罗 著. 复杂——诞生于秩序与混沌边缘的科学. 陈玲 译. 北京：生活·读书·新知 三联书店，1998

[25] 约翰·H·霍兰 著. 隐秩序——适应性造就复杂性. 周晓牧，韩晖 译. 上海：上海科技教育出版社，

2000
[26] 刘勇，康立山 等. 非数值并行计算(第 2 册)——遗传算法. 北京：科学出版社，1995.
[27] 蔡自兴，徐光佑. 人工智能及其应用(第 2 版). 北京：清华大学出版社，1996
[28] 史忠植. 高级人工智能. 北京：科学出版社，1998
[29] 刘健勤. 人工生命理论及其应用. 北京：冶金工业出版社，1997
[30] 周明，孙树栋. 遗传算法原理及其应用. 北京：国防工业出版社，1999
[31] 陈国良，王煦法 等. 遗传算法及其应用. 北京：人民邮电出版社，1996
[32] 成思危. 复杂性科学探索. 北京：民主与建设出版社，1999
[33] 中国科学院《复杂性研究》编委会. 复杂性研究. 北京：科学出版社，1993
[34] 李国杰. 世纪电脑. 北京：科学技术文献出版社，1999
[35] 高济. 基于知识的软件智能化技术. 杭州：浙江大学出版社，2000
[36] 朱淼良，杨建刚，吴春明. 自主式智能系统. 杭州：浙江大学出版社，2000
[37] 赵南元. 认知科学与广义进化论. 北京：清华大学出版社，1994
[38] 许国志. 系统科学. 上海：上海科技教育出版社，2000
[39] 许国志. 系统科学与工程研究. 上海：上海科技教育出版社，2000
[40] 潘正军，康立山 等. 演化计算. 清华大学出版社，广西科学技术出版社，1998
[41] 孙增圻 等. 智能控制理论与技术. 清华大学出版社，广西科学技术出版社，1998
[42] 陈阅增. 普通生物学——生命科学通论. 北京：高等教育出版社，1997
[43] 张昀. 生物进化. 北京：北京大学出版社，1996
[44] 李衍达. 信息科学与生物之谜. 世界科技研究与发展，1999，23(3)：26～30
[45] 丁承民，张传生 等. 遗传算法纵横谈. 信息与控制，1997，26(1)：40～46
[46] 贺前华，韦岗 等. 基因算法研究进展. 电子学报，1998，26(10)：118～122
[47] 段玉倩，贺家李. 遗传算法及其改进. 电力系统及其自动化学报，1998，10(1)：39～51
[48] 张东摩，李红兵. 人工智能研究动态与发展趋势. 计算机科学，1998，25(2)：5～8
[49] 周水林，潘云鹤. 从智能模拟到智能工程——论人工智能研究范式的转变. 计算机科学，1999，26(7)：18～22

第 2 章　基本遗传算法

本章从借助遗传算法优化一个简单函数的实例入手,分析遗传算法的基本特征。其次,我们将介绍和比较更多的遗传操作的算法。最后讨论一下遗传算法设计的若干问题。

2.1　简单函数优化的实例

考虑下列一元函数求最大值的优化问题:

$$f(x) = x\sin(10\pi \cdot x) + 2.0 \qquad x \in [-1,2] \tag{2.1}$$

该函数曲线如图 2.1 所示。由于 $f(x)$ 在区间 $[-1,2]$ 可微,我们首先用微分法求取 $f(x)$ 的最大值。

$$f'(x) = \sin(10\pi \cdot x) + 10\pi \cdot x \cdot \cos(10\pi \cdot x) = 0 \tag{2.2}$$

即

$$\tan(10\pi \cdot x) = -10\pi \cdot x \tag{2.3}$$

上式的解有无穷多个:

$$\begin{cases} x_i = \dfrac{2i-1}{20} + \varepsilon_i, \quad i = 1,2,\cdots & (2.4) \\ x_0 = 0 & (2.5) \\ x_i = \dfrac{2i+1}{20} + \varepsilon_i, \quad i = -1,-2,\cdots & (2.6) \end{cases}$$

式中 ε_i($i=1,2,\cdots$ 及 $i=-1,-2,\cdots$)是一接近于 0 的实数递减序列。

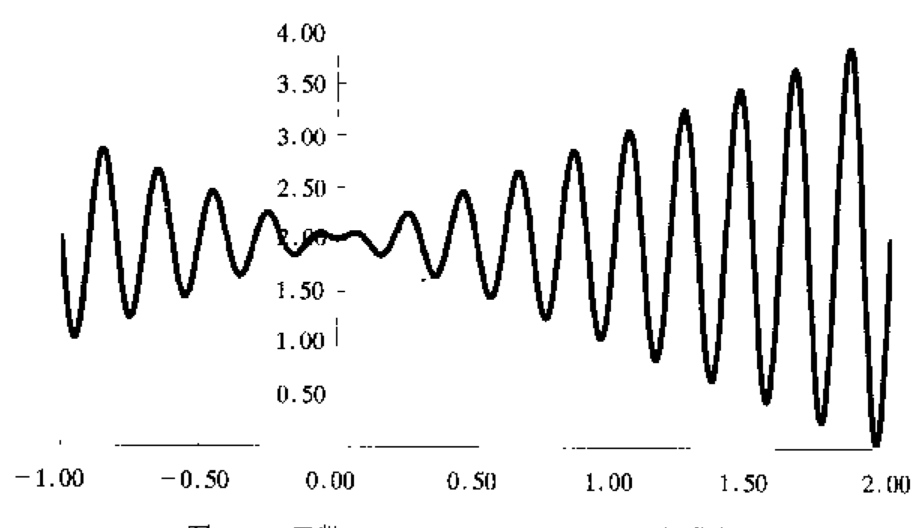


图 2.1　函数 $f(x) = x\sin(10\pi x) + 2.0$ 的曲线

当 i 为奇数时 x_i 对应局部极大值点,i 为偶数时 x_i 对应局部极小值点。显然 x_{19} 即为区间 $[-1,2]$ 内的最大值点:

$$x_{19} = \frac{37}{20} + \varepsilon_{19} = 1.85 + \varepsilon_{19} \tag{2.7}$$

此时，函数最大值 $f(x_{19})$ 比 $f(1.85)=3.85$ 稍大。

下面我们用遗传算法解决上述简单函数的最优化问题。

(1) **编码** 变量 x 作为实数，可以视为遗传算法的表现型形式。从表现型到基因型的映射称为编码。通常采用二进制编码形式，将某个变量值代表的个体表示为一个{0,1}二进制串。当然，串长取决于求解的精度。如果设定求解精确到6位小数，由于区间长度为 $2-(-1)=3$，必须将闭区间 $[-1,2]$ 分为 3×10^6 等份。因为

$$2\ 097\ 152 = 2^{21} < 3 \times 10^6 \leqslant 2^{22} = 4\ 194\ 304$$

所以编码的二进制串长至少需要22位。

将一个二进制串 $(b_{21}b_{20}\cdots b_0)$ 转化为区间 $[-1,2]$ 内对应的实数值很简单，只需要采取以下两步：

① 将一个二进制串 $(b_{21}b_{20}\cdots b_0)$ 代表的二进制数化为10进制数：

$$(b_{21}b_{20}\cdots b_0)_2 = \left(\sum_{i=0}^{21} b_i \cdot 2^i\right)_{10} = x' \tag{2.8}$$

② x' 对应的区间 $[-1,2]$ 内的实数：

$$x = -1.0 + x' \cdot \frac{2-(-1)}{2^{22}-1} \tag{2.9}$$

例如，一个二进制串 $s1=<1000101110110101000111>$ 表示实数值0.637 197。

$x'=(1000101110110101000111)_2=2\ 288\ 967$

$x=-1.0+2\ 288\ 967\cdot\frac{3}{2^{22}-1}=0.637\ 197$

二进制串＜0000000000000000000000＞与＜1111111111111111111111＞，则分别表示区间的两个端点值 -1 和2。

(2) **产生初始种群** 一个个体由串长为22的随机产生的二进制串组成染色体的基因码，我们可以产生一定数目的个体组成种群，种群的大小(规模)就是指种群中的个体数目。

(3) **计算适应度** 对于个体的适应度计算，考虑到本例目标函数在定义域内均大于0，而且是求函数最大值，所以直接引用目标函数作为适应度函数：

$$f(s) = f(x) \tag{2.10}$$

这里二进制串 s 对应变量 x 的值。

例如，有三个个体的二进制串为：

$$s_1 = < 1000101110110101000111 >$$
$$s_2 = < 0000001110000000010000 >$$
$$s_3 = < 1110000000111111000101 >$$

分别对应于变量值 $x_1=0.637\ 197$，$x_2=-0.958\ 973$，$x_3=1.627\ 888$，个体的适应度计算如下：

$$f(s_1) = f(x_1) = 2.586\ 345$$
$$f(s_2) = f(x_2) = 1.078\ 878$$
$$f(s_3) = f(x_3) = 3.250\ 650$$

显然,三个个体中 s_3 的适应度最大, s_3 为最佳个体。

(4) **遗传操作** 这里介绍交叉和变异这两个遗传操作是如何工作的。

下面是经过选择操作(第 1 章中介绍过的轮盘赌选择方法)的两个个体,首先执行单点交叉,如

$$s_2 = < 00000 \mid 01110000000010000 >$$

$$s_3 = < 11100 \mid 00000111111000101 >$$

随机选择一个交叉点,例如第 5 位与第 6 位之间的位置,交叉后产生新的子个体:

$$s_2' = < 00000 \mid 00000111111000101 >$$

$$s_3' = < 11100 \mid 01110000000010000 >$$

这两个子个体的适应度分别为:

$$f(s_2') = f(-0.998\ 113) = 1.940\ 865$$

$$f(s_3') = f(1.666\ 028) = 3.459\ 245$$

我们注意到,个体 s_3' 的适应度比其两个父个体的适应度高。

下面考察变异操作。假设已经以一小概率选择了 s_3 的第 5 个遗传因子(即第 5 位)变异,遗传因子由原来的 0 变为 1,产生新的个体为 $s_3' = < 1110100000111111000101 >$,计算该个体的适应度: $f(s_3') = f(1.721\ 638) = 0.917\ 743$,发现个体 s_3' 的适应度比其父个体的适应度减少了,但如果选择第 10 个遗传因子变异,产生新的个体为 $s_3'' = < 1110000001111111000101 >$, $f(s_3'') = f(1.630\ 818) = 3.343\ 555$,又发现个体 s_3'' 的适应度比其父个体的适应度改善了。这说明了变异操作的“扰动”作用。

(5) **模拟结果** 设定种群大小为 50,交叉概率 $p_c = 0.25$,变异概率 $p_m = 0.01$,按照上述的基本遗传算法(Simple Genetic Algorithm, SGA),在运行到 89 代时获得最佳个体:

$$S_{max} = < 1101001111110011001111 >$$

$$x_{max} = 1.850\ 549, \qquad f(x_{max}) = 3.850\ 274$$

这个个体对应的解与微分法预计最优解的情况吻合,最大函数值比 3.85 略大,可以作为问题的近似最优解。表 2.1 列出了模拟世代的种群中最佳个体的演变情况。

表 2.1 模拟世代的种群中最佳个体的演变情况(150 代终止)

世代数	个体的二进制串	x	适应度
1	1000111000010110001111	1.831 624	3.534 806
4	0000011011000101001111	1.842 416	3.790 362
7	1110101011100111001111	1.854 860	3.833 280
11	0110101011100111001111	1.854 860	3.833 286
17	1110101011111101001111	1.847 536	3.842 004
18	0000111011111101001111	1.847 554	3.842 102
34	1100001101111011001111	1.853 290	3.843 402
40	1101001000100011001111	1.848 443	3.846 232
54	1000110110100011001111	1.848 699	3.847 155
71	0100110110001011001111	1.850 897	3.850 162
89	1101001111110011001111	1.850 549	3.850 274
150	1101001111110011001111	1.850 549	3.850 274

从上述简单函数优化的实例中可以看出，遗传算法是一种强大的随机搜索方法。由于数学函数优化问题不需要专门领域的知识，且能较好地反映算法本身的实际效能，所以常用于 GA 的测试问题。下面四个为具有相当复杂度的常用测试函数：

① Shubert 函数

$$f(x,y) = \left\{\sum_{i=1}^{5} i\cos[(i+1)x+1]\right\}\left\{\sum_{i=1}^{5} i\cos[(i+1)y+1]\right\} + 0.5[(x+1.425\,13)^2+(y+0.800\,32)^2] \tag{2.11}$$

此函数有 760 个局部极小点，其中只有一个 $(-1.425\,13, -0.800\,32)$ 为全局最小，最小值为 186.730 9。自变量取值范围 $-10<x, y<10$。此函数极易陷入局部极小值 186.34。

② Camel 函数

$$f(x,y) = (4-2.1x^2+\frac{x^4}{3})x^2 + xy + (-4+4y^2)y^2 \tag{2.12}$$

此函数有 6 个局部极小点，其中有两个 $(-0.089\,8, 0.712\,6)$ 和 $(0.089\,8, -0.712\,6)$ 为全局最小点，最小值为 $-1.031\,628$，自变量的取值范围为 $-100<x, y<100$。

③ De Jones's F_5(Shekel's Foxholes)函数

$$f_5(x_i) = 0.02 + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6} \tag{2.13}$$

其中 $(a_{ij})_{2\times 25} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$。自变量取值范围为 $-65\,536<x_i<65\,536$。此函数有多个局部极大点，一般其函数值大于 1 即可认为收敛。

④ Shaffer's F_6 函数

$$f(x,y) = 0.5 - \frac{\sin^2\sqrt{x^2+y^2}-0.5}{(1+0.001(x^2+y^2))^2} \tag{2.14}$$

此函数有无限个局部极大点，其中只有一个 $(0,0)$ 为全局最大，最大值为 1。自变量的取值范围为 $-100<x, y<100$。此函数最大值峰周围有一个圈脊，它们的取值均为 0.990 283，因此很容易停滞在此局部极大点。

除了以上四个函数之外，常用的测试函数还有 De Jones's $F_1 \sim F_4$ 等。

2.2 遗传基因型

Holland 提出的遗传算法是采用二进制编码来表现个体的遗传基因型的，它使用的编码符号集由二进制符号 0 和 1 组成的，因此实际的遗传基因型是一个二进制符号串，其优点在于编、解码操作简单，交叉、变异等遗传操作便于实现，而且便于利用模式定理进行理论分析等；其缺点在于，不便于反映所求问题的特定知识，对于一些连续函数的优化问题等，也由于遗传算法的随机特性而使得其局部搜索能力较差，对于一些多维、高精度要求的连续函数优化，二进制编码存在着连续函数离散化时的映射误差，个体编码串较短时，可能达不到精度要求；而

个体编码串的长度较长时，虽然能提高精度，但却会使算法的搜索空间急剧扩大。显然，如果个体编码串特别长时，会造成遗传算法的性能降低。后来许多学者对遗传算法的编码方法进行了多种改进，例如，为提高遗传算法局部搜索能力，提出了格雷码(Grey Code)编码；为改善遗传算法的计算复杂性、提高运算效率，提出了浮点数编码、符号编码方法等。为便于利用求解问题的专门知识，便于相关近似算法之间的混合使用，提出了符号编码法；此外还有多参数级联编码和交叉编码方法，近年来，随着生物计算理论研究的兴起，有人提出DNA编码法(详见9.4节)，并在模糊控制器优化的应用中取得很好的效果。

我们知道，遗传算法中进化过程是建立在编码机制基础上的，编码对于算法的性能如搜索能力和种群多样性等影响很大。就二进制编码和浮点数编码比较而言，一般二进制编码比浮点数编码搜索能力强，但浮点数编码比二进制编码在变异操作上能够保持更好的种群多样性。

下面就常用的二进制编码和浮点数编码的表示机制进行分析和比较。首先讨论一下标准的交叉操作的情况。

1. 浮点数编码

假设* 假定种群中个体数目为 n，x_t^i 表示第 t 代的第 i 个个体，$i\in\{1,2,\cdots,n\}$。每个个体的基因位数 $L=m$，由 m 个实数构成，个体 $\boldsymbol{x}_t^i\in\mathbf{R}^m$，$\boldsymbol{x}_t^i$ 可以表示 m 维的行向量，即 $\boldsymbol{x}_t^i=(x_t^{i(1)}\ x_t^{i(2)}\ \cdots\ x_t^{i(m)})$。这样第 t 代的种群 $\mathbf{X}_t$ 可以表示为 $n\times m$ 的矩阵，$\mathbf{X}_t=(\boldsymbol{x}_t^1\ \boldsymbol{x}_t^2\ \cdots\ \boldsymbol{x}_t^n)^{\mathrm{T}}$。初始种群矩阵 $\mathbf{X}_0=(\boldsymbol{x}_0^1\ \ \boldsymbol{x}_0^2\ \ \cdots\ \ \boldsymbol{x}_0^n)^{\mathrm{T}}$ 中没有相同的两行，而且每列中没有相同的元素，即种群中所有个体互异，对任意 $i\neq j\ (i,j\in\{1,2,\cdots,n\})$ 有 $\boldsymbol{x}_0^i\neq\boldsymbol{x}_0^j$；而且个体中没有两个个体的同一基因位是相同的，即对任意 $i\neq j\ (i,j\in\{1,2,\cdots,n\},k\in\{1,2,\cdots,m\})$ 有 $x_0^{i(k)}\neq x_0^{j(k)}$。

定理 2.1 在假设*的情况下，设初始种群经交叉操作产生的新个体属于可数集合 D，则集合 D 中的元素的数目 T_D 为

$$T_D=2(m-1)C_n^2 \tag{2.15}$$

证明 因为种群的个体数目为 n，任意取两个个体进行交叉的可能情况为 C_n^2，又知有$(m-1)$个交叉位置，由假设条件保证交叉后产生的两个新个体是不相同的，因此所有这些可能情况产生的个体构成集合 D，所以定理2.1成立。

定理 2.2 设两个互异的个体 x_t^i 和x_t^j，且 $x_t^{i(k)}-x_t^{j(k)}=\delta$，那么它们交叉后产生的新个体 x'_t 满足

$$\left|x'^{(k)}_t-x_t^{p(k)}\right|=\delta \tag{2.16}$$

或者满足

$$\left|x'^{(k)}_t-x_t^{p(k)}\right|=0 \tag{2.17}$$

其中 $i\neq j\ (i,j\in\{1,2,\cdots,n\},k\in\{1,2,\cdots,m\},p\in\{i,j\})$。

证明 因为交叉后产生的新个体 x'_t 的第k 个基因位$x_t'^{(k)}\in\{x_t^{i(k)},x_t^{j(k)}\}$，所以定理2.2显然成立。

定理 2.2 表明了对于任意两个互异的个体，交叉后产生的新个体一定在这两个父个体所构成的超体 $\prod\limits_{k=1}^{m}[\min(x_i^{(k)},x_j^{(k)}),\max(x_i^{(k)},x_j^{(k)})]$的顶点上。

2. 二进制编码

假设种群中个体数目为 n, $\boldsymbol{x}_t^i$ 表示第 t 代的第 i 个个体, $i \in \{1,2,\cdots,n\}$。每个个体用 l 位二进制表示。这样每个个体 $\boldsymbol{x}_t^i \in \{IB\}^{ml}$, $IB \in \{0,1\}$,这样每个个体基因位数目 $L = ml$。个体 $\boldsymbol{x}_t^i$ 可以表示为 ml 维的行向量,即 $\boldsymbol{x}_t^i = [x_t^{i(1)} \cdots x_t^{i(l)}\ x_t^{i(l+1)} \cdots x_t^{i(2l)} \cdots x_t^{i((m-1)l+1)} \cdots x_t^{i(ml)}]$。第 t 代种群 $\boldsymbol{X}_t$ 可以表示为一个 $n \times ml$ 的矩阵 $\boldsymbol{X}_t = [\boldsymbol{x}_t^1\ \boldsymbol{x}_t^2\ \cdots\ \boldsymbol{x}_t^n]^{\mathrm{T}}$。个体 $\boldsymbol{x}_t^i$ 的第 k 个长度为 l 的二进制码串转化为实数的解码函数 Γ 为

$$\Gamma(x_t^i, k) = u_k + \frac{v_k - u_k}{2^l - 1}\left(\sum_{j=1}^{l} x_t^{i(kl+j)} \times 2^{j-1}\right) \tag{2.18}$$

式中 v_k 和 u_k 分别为第 k 个实数范围的上限和下限。

定理 2.3　设用二进制编码的初始种群 X_0 是假定 * 的浮点数编码的初始种群转化为相应二进制编码后的种群。设经交叉操作产生的新的个体属于可数集合 B,则集合 B 中元素的数目 T_B 满足:

$$T_B \geqslant 2n_0(n - n_0) + T_D \tag{2.19}$$

这里 n_0 表示种群中第一位二进制码为 0 的个体数目。

证明　当交叉位置为 $l, 2l, \cdots, (m-1)l$ 时,可能产生的不同的新的个体的数目与浮点数编码相同为 T_D,当交叉位置为 1 时,将 n 个个体分为第一位为 0 和第一位为 1 两组,分别记作 B_0 和 B_1,第一位为 0 的个体数目为 n_0,则第一位为 1 的个体数目为 $n - n_0$。那么容易证明从 B_0 中选出一个个体与从 B_1 中选出一个个体交叉产生的新个体的可能情况是 $2n_0(n - n_0)$,而且不与交叉位置 $l, 2l, \cdots, (m-1)l$ 的情况相重复。交叉位置还可以选其他位置,所以 $T_B \geqslant 2n_0(n - n_0) + T_D$ 成立。

定理 2.3 表明只要 $n_0 \neq 0$,或者 $n_0 \neq n$,用二进制编码交叉可产生的遍历搜索空间的不同个体的数目大于用浮点数编码的情况。也就是说,就交叉操作而言,二进制编码的搜索能力比浮点数的搜索能力强。由(2.19)式可以看出种群的个体数目越大,二进制编码的搜索空间的能力比浮点数的搜索能力强就体现得越充分。

定理 2.4　若两个个体 x_t^i, x_t^j,对一给定的 $k \in \{1,2,\cdots,m\}$,满足

$$\Gamma(x_t^i, k) - \Gamma(x_t^j, k) = \frac{v_k - u_k}{2^l - 1}(2^p - 2^q) \tag{2.20}$$

其中 $p, q \in \{1,2,\cdots,l-1\}$, $q > p$。则 x_t^i, x_t^j 经交叉后产生的两个新个体 $(x_t^i)', (x_t^j)'$ 统称为 x_t',满足

$$\Gamma(x_t^i, k) < \Gamma(x'_t, k) < \Gamma(x_t^j, k) \tag{2.21}$$

的概率不为 0。(证明从略)

定理 2.5　若两个个体 x_t^i, x_t^j,对一给定的 $k \in \{1,2,\cdots,m\}$,满足:

$$\Gamma(x_t^i, k) - \Gamma(x_t^j, k) = \frac{v_k - u_k}{2^l - 1}(2^p - 2^q) \tag{2.22}$$

其中 $p, q \in \{1,2,\cdots,l-1\}$, $q > p$。则 x_t^i, x_t^j 经交叉后产生的两个新个体 $(x_t^i)', (x_t^j)'$,满足

$$\Gamma(x_t^i, k) < \Gamma((x_t^i)', k) \tag{2.23}$$

和

$$\Gamma(x_t^j, k) < \Gamma((x_t^j)', k) \tag{2.24}$$

的概率不为 0(证明从略)。

由定理 2.4 和定理 2.5 可以看出,用二进制编码,交叉操作产生得到的新个体既可能在其对应的二进制数所构成的超体 $\prod_{k=1}^{m}[\min(\Gamma(x_t^i,k),\Gamma(x_t^j,k)),\max(\Gamma(x_t^i,k),\Gamma(x_t^j,k))]$的内部,也可能在其外部。而由浮点数编码由定理 2.2 可知,交叉后产生的新个体只能在其父个体构成的超体的顶点上。

通过以上分析,可以得出结论:在使用交叉操作时,二进制编码比浮点数编码产生新个体的可能情况多,而且产生的新个体不受父个体所构成的超体的限制。总之,二进制编码比浮点数编码的搜索能力强,而且随着种群大小的增大,这种差别就越明显。

下面我们分析一下变异操作的情况。

浮点数编码对于个体 x_t^i 的第 p 个基因进行变异操作 ψ_D 定义如下:

$$\psi_D(x_t^i, p) = x_t^{i(p)} + N(0, \delta) \tag{2.25}$$

其中 $N(0,\delta)$是均值为 0、方差为 δ 的高斯噪声。可见浮点数编码操作的变异量可以任意地小。这样可以保证只要变异量足够小,产生的新个体可以与父个体充分地接近。

对于二进制编码,对个体的 x_t^i 的第 p 个基因进行变异操作 ψ_B 定义如下:

$$\psi_B(x_t^i, p) = \begin{cases} 0, & x_t^{i(p)} = 1 \\ 1, & x_t^{i(p)} = 0 \end{cases} \tag{2.26}$$

定理 2.6 对于任意给定的 $i\in\{1,2,\cdots,n\}$, $k\in\{1,2,\cdots,m\}$,二进制编码个体 x_t^i 所对应的第k 个实数的变异最小量为$\frac{v_k-u_k}{2^l-1}$。

定理 2.6 说明对于二进制编码,变异的最小量不能任意小,它受到编码长度的限制,l 越大,变异的最小量就越小。这样,即使在最优解附近,由变异操作也可能遍历不到最优解。

定理 2.7 对于任意给定 $i\in\{1,2,\cdots,n\}$, $k\in\{1,2,\cdots,m\}$,二进制编码个体 x_t^i 只变异一位,产生新个体 x'_t,设 $s^k=|\Gamma(x_t^i,k)-\Gamma(x'_t,k)|$,则 s^k 的最大值为:

$$s_{\max}^k = \frac{v_k - u_k}{2^l - 1} \times 2^{l-1} \tag{2.27}$$

s^k 的最小值为:

$$s_{\min}^k = \frac{v_k - u_k}{2^l - 1} \tag{2.28}$$

定理 2.7 说明了对二进制编码,变异操作不能保证父个体与新个体的充分接近。由定理 2.6 和 2.7 可以得出以下结论:二进制编码的种群稳定性比浮点数编码差。

理论上而言,编码应该适合要解决的问题,而不是简单的描述问题。Balakrishman 等较全面地讨论了不同编码方法的一组特性,针对一类特别的应用,为设计和选择编码方法提供了指南,主要有以下九个特性:

(1) **完全性**(completeness) 原则上,分布在所有问题域的解都可能被构造出来。

(2) **封闭性**(closure) 每个基因编码对应一个可接受的个体,封闭性保证系统从不产生无效的个体。

(3) **紧致性**(compactness) 若两种基因编码 g_1 和 g_2 都被解码成相同的个体,若 g_1 比 g_2 占的空间少,就认为 g_1 比 g_2 紧致。

(4) **可扩展性**(scalability)　对于具体的问题,编码的大小确定了解码的时间,两者存在一定的函数关系,若增加一种表现型,作为基因型的编码大小也作出相应的增加。

(5) **多重性**(multiplicity)　多个基因型解码成一个表现型,即从基因型到相应的表现型空间是多对一的关系,这是基因的多重性。若相同的基因型被解码成不同的表现型,这是表现型多重性。

(6) **个体可塑性**(flexibility)　决定表现型与相应给定基因型是受环境影响的。

(7) **模块性**(modularity)　若表现型的构成中有多个重复的结构,在基因型编码中这种重复是应当避免的。

(8) **冗余性**(redundancy)　冗余性能够提高可靠性和鲁棒性(robustness)。

(9) **复杂性**(complexity)　包括基因型的结构复杂性,解码复杂性,计算时空复杂性(基因解码、适应值、再生等)。

其中满意的特性是:完全性、可测性和复杂性。但以上特性有时是矛盾的。

2.3　适应度函数及其尺度变换

遗传算法在进化搜索中基本不利用外部信息,仅以适应度函数(fitness function)为依据,利用种群中每个个体的适应度值来进行搜索。因此适应度函数的选取至关重要,直接影响到遗传算法的收敛速度以及能否找到最优解。一般而言,适应度函数是由目标函数变换而成的。对目标函数值域的某种映射变换称为适应度的尺度变换(fitness scaling)。

2.3.1　几种常见的适应度函数

适应度函数基本上有以下三种:

① 直接以待求解的目标函数的转化为适应度函数,即:

$$\text{若目标函数为最大化问题}\quad \mathrm{Fit}(f(x)) = f(x) \tag{2.29}$$

$$\text{若目标函数为最小问题}\quad \mathrm{Fit}(f(x)) = -f(x) \tag{2.30}$$

这种适应度函数简单直观,但存在两个问题,其一是可能不满足常用的轮盘赌选择中概率非负的要求;其二是某些代求解的函数在函数值分布上相差很大,由此得到的平均适应度可能不利于体现种群的平均性能,影响算法的性能。

② 若目标函数为最小问题,则

$$\mathrm{Fit}(f(x)) = \begin{cases} c_{\max} - f(x), & f(x) < c_{\max} \\ 0, & \text{其他} \end{cases} \tag{2.31}$$

式中 $c_{\max}$为 $f(x)$的最大值估计。

若目标函数为最大问题,则

$$\mathrm{Fit}(f(x)) = \begin{cases} f(x) - c_{\min}, & f(x) > c_{\min} \\ 0, & \text{其他} \end{cases} \tag{2.32}$$

式中 $c_{\min}$为 $f(x)$的最小值估计。

这种方法是对第一种方法的改进,可以称为“界限构造法”,但有时存在界限值预先估计困难、不可能精确的问题。

③ 若目标函数为最小问题

$$\text{Fit}(f(x)) = \frac{1}{1+c+f(x)} \qquad c \geqslant 0,\ c+f(x) \geqslant 0 \tag{2.33}$$

若目标函数为最大问题

$$\text{Fit}(f(x)) = \frac{1}{1+c-f(x)} \qquad c \geqslant 0,\ c-f(x) \geqslant 0 \tag{2.34}$$

这种方法与第二种方法类似，c 为目标函数界限的保守估计值。

2.3.2 适应度函数的作用

在选择操作时会出现以下问题：

① 在遗传进化的初期，通常会产生一些超常的个体，若按照比例选择法，这些异常个体因竞争力太突出而控制了选择过程，影响算法的全局优化性能。

② 在遗传进化的后期，即算法接近收敛时，由于种群中个体适应度差异较小时，继续优化的潜能降低，可能获得某个局部最优解。

上述问题，我们通常称为遗传算法的欺骗问题。适应度函数设计不当有可能造成这种问题的出现。

设 n 个个体的适应度函数值 $F_1 \leqslant F_2 \leqslant \cdots \leqslant F_n$，记

$$F_{i+1} - F_i = \Delta_i \qquad (i = 1, 2, \cdots, n-1) \tag{2.35}$$

若 $\Delta = \max\{\Delta_i \mid (i=1,2,\cdots,n-2)\}$，且 $\Delta_{n-1} \geqslant \frac{(n-2)(n-1)\Delta}{2}$，则 $F_{n-1} \leqslant \frac{1}{n}\sum_{i=1}^{n} F_i \leqslant F_n$。

若 $\Delta = \max\{\Delta_i \mid (i=2,3,\cdots,n-2)\}$，且 $\Delta_1 \geqslant \frac{(n-2)(n-1)\Delta}{2}$，则 $F_1 \leqslant \frac{1}{n}\sum_{i=1}^{n} F_i \leqslant F_2$。

上面两种情况下，适应度函数值的分布对遗传搜索而言是极不合理的，所以适应度函数的设计是遗传算法设计的一个重要方面。

2.3.3 适应度函数的设计

适应度函数设计主要满足以下条件：

(1) **单值、连续、非负、最大化** 这个条件是很容易理解和实现的。

(2) **合理、一致性** 要求适应度值反映对应解的优劣程度，这个条件的达成往往比较难以衡量。

(3) **计算量小** 适应度函数设计应尽可能简单，这样可以减少计算时间和空间上的复杂性，降低计算成本。

(4) **通用性强** 适应度对某类具体问题，应尽可能通用，最好无需使用者改变适应度函数中的参数。从目前而言，这个条件应该是不属于强要求。

在第 6 章以后的章节里，我们将根据遗传算法的实际应用，再来讨论这个问题。

2.3.4 适应度函数的尺度变换

常用的尺度变换方法有以下几种：

1. 线性变换法

假设原适应度函数为 f，变换后的适应度函数为 f'，则线性变换可用下式表示：

$$f' = \alpha * f + \beta \tag{2.36}$$

上式中的系数确定方法有多种，但要满足以下条件：

① 原适应度的平均值要等于定标后的适应度平均值，以保证适应度为平均值的个体在下

一代的期望复制数为 1，即：

$$f'_{avg} = f_{avg} \tag{2.37}$$

② 变换后的适应度最大值应等于原适应度平均值的指定倍数，以控制适应度最大的个体在下一代中的复制数。试验表明，指定倍数 c_{mult} 可在 1.0～2.0 范围内。即根据上述条件可确定线性比例的系数：

$$f'_{max} = c_{mult} f_{avg} \tag{2.38}$$

$$\alpha = \frac{(c_{mult} - 1) f_{avg}}{f_{max} - f_{avg}}, \qquad \beta = \frac{(f_{max} - c_{mult} f_{avg}) f_{avg}}{f_{max} - f_{avg}} \tag{2.39}$$

如图 2.2 所示，线性变换法变换了适应度之间的差距，保持了种群内的多样性，并且计算简便，易于实现。如果种群内某些个体适应度远远低于平均值时，有可能出现变换后适应度值为负的情况，为此，考虑到保证最小适应度值非负的条件，进行如下的变换：

$$\alpha = \frac{f_{avg}}{f_{avg} - f_{min}}, \qquad \beta = \frac{-f_{min} f_{avg}}{f_{avg} - f_{min}} \tag{2.40}$$

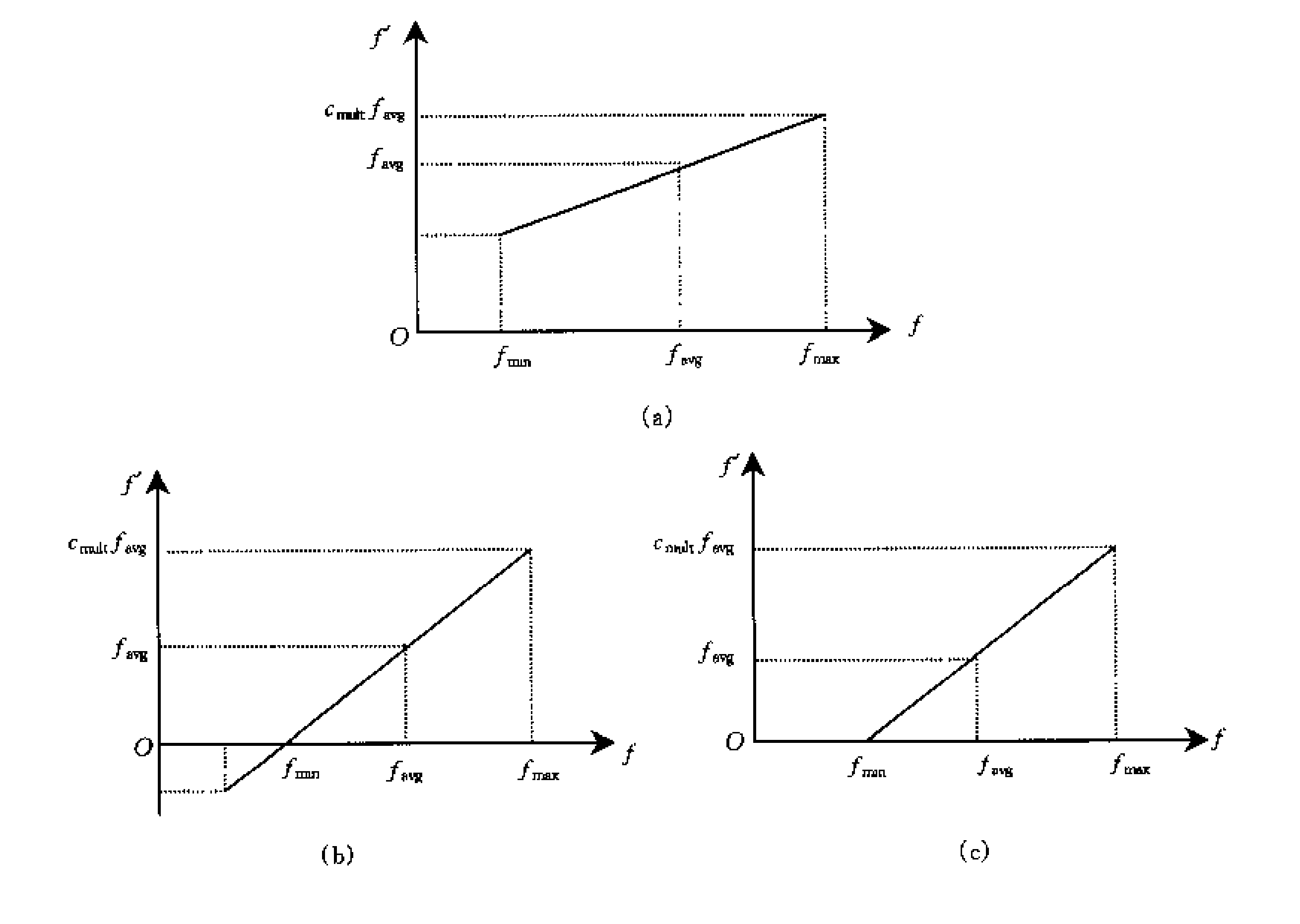


图 2.2　适应度函数的线性调整

(a) 正常情况；(b) 适应度出现负值；(c) 适应度出现负值时的变换

2. 幂函数变换法

变换公式为：

$$f' = f^k \tag{2.41}$$

上式中的幂指数 k 与所求的最优化问题有关，结合一些试验进行一定程度的精细变换才

能获得较好的结果。

3. 指数变换法

变换公式为：

$$f' = e^{-af} \tag{2.42}$$

这种变换方法的基本思想来源于模拟退火过程(simulated annealing, SA)，其中的系数 a 决定了复制的强制性，其值越小，复制的强制就越趋向于那些具有最大适应度的个体。

2.4 遗传操作

在第1章中，我们已经熟悉了几个简单遗传操作方法，例如轮盘赌选择、二进制编码的单点交叉和变异。本节将介绍常用的遗传操作方法，并进行比较。

2.4.1 选择(selection)

选择过程的第一步是计算适应度。在被选集中每个个体具有一个选择概率，这个选择概率取决于种群中个体的适应度及其分布。

下面一些概念可以用来比较不同的选择算法：

(1) **选择压力(selection pressure)** 最佳个体选中的概率与平均选中概率的比值。

(2) **偏差(bias)** 个体正规化适应度与其期望再生概率的绝对差值。

(3) **个体扩展(spread)** 单个个体子代个数的范围。

(4) **多样化损失(loss of diversity)** 在选择阶段未选中个体数目占种群的比例。

(5) **选择强度(selection intensity)** 将正规高斯分布应用于选择方法，期望平均适应度。

(6) **选择方差(selection variance)** 将正规高斯分布应用于选择方法，期望种群适应度的方差。

个体选择概率的常用分配方法有以下两种：

(1) **按比例的适应度分配(proportional fitness assignment)** 按比例的适应度分配，可称为选择的蒙特卡罗法，是利用比例于各个个体适应度的概率决定其子孙的遗留可能性。若某个个体 i，其适应度为 f_i，则其被选取的概率表示为

$$P_i = \frac{f_i}{\sum_{i=1}^{M} f_i} \tag{2.43}$$

表2.2给出了按比例的适应度计算方法的例子。表中采用的比率 $k=2$，当选择的概率给定后，产生[0, 1]区间的均匀随机数来决定哪个个体参加交配。显然选择概率大的个体，能多次被选中，它的遗传因子就会在种群中扩大。

表2.2 个体选择概率计算

个体	f	f^2	P_i
1	2.5	6.25	0.18
2	1.0	1.00	0.03
3	3.0	9.00	0.26
4	1.2	1.44	0.04
5	2.1	4.41	0.13
6	0.8	0.64	0.02
7	2.5	6.25	0.18
8	1.3	1.69	0.05
9	0.9	0.81	0.02
10	1.8	3.24	0.09

(2) **基于排序的适应度分配(rank-based fitness assignment)** 在基于排序的适应度分配中，种群按目标值进行排序。适应度仅仅取决于个体在种群中的序位，而不是实际的目

标值。排序方法克服了比例适应度计算的尺度问题,当选择压力太小的情况下,以及选择导致搜索带迅速变窄而产生的过早收敛。因此,再生范围被局限。排序方法引入种群均匀尺度,提供了控制选择压力的简单有效的方法。

排序方法比比例方法表现出更好的鲁棒性,因此,不失为一种好的选择方法。

设定 N 为种群大小,Pos 为个体在种群中的序位,SP 为选择压力,个体的适应度可以计算如下:

线形排序:

$$\mathrm{Fit}(Pos) = 2 - SP + \frac{2(SP-1)(Pos-1)}{N-1} \qquad SP \in [1.0, 2.0] \tag{2.44}$$

非线形排序:

$$\mathrm{Fit}(Pos) = \frac{NX^{Pos-1}}{\sum_{i=1}^{N} X^{i-1}} \tag{2.45}$$

其中 X 是下列多项式方程的根:

$$(SP-1)\cdot X^{N-1} + SP\cdot X^{N-2} + \cdots + SP\cdot X + SP = 0 \tag{2.46}$$

$$SP \in [1.0, N-2.0]$$

选择强度:

$$SelInt_{\mathrm{Rank}} = \frac{SP-1}{\sqrt{\pi}} \tag{2.47}$$

多样化损失:

$$LossDiv_{\mathrm{Rank}} = (SP-1)/4 \tag{2.48}$$

选择方差:

$$SelVar_{\mathrm{Rank}}(SP) = 1 - ((SP-1)^2/\pi) = 1 - SelInt_{\mathrm{Rank}} SP^2 \tag{2.49}$$

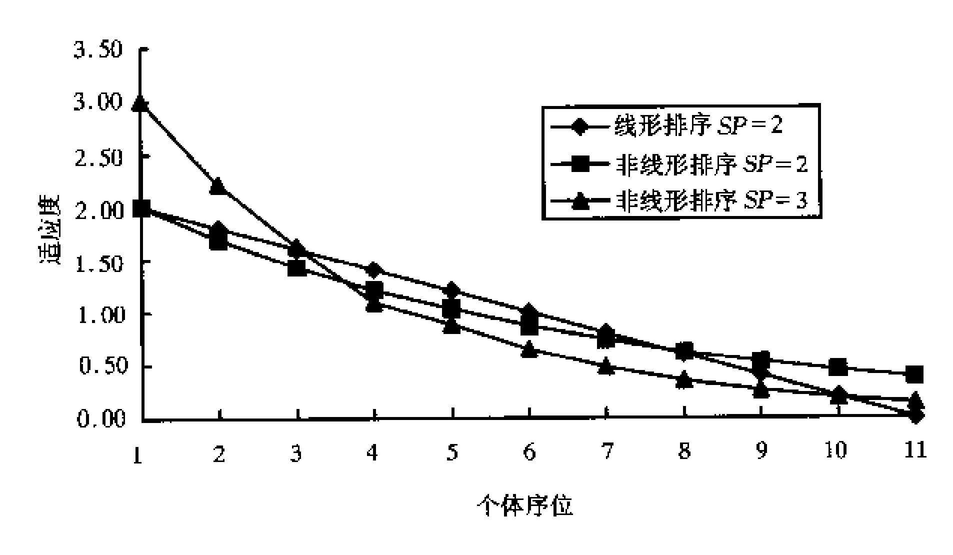


图 2.3　基于排序的适应度计算

关于个体的选择概率计算,Baker 提出了线性排序的计算公式:

$$p_i = \frac{1}{N}\left[\eta^+ - (\eta^+ - \eta^-)\frac{i-1}{N-1}\right] \tag{2.50}$$

上式中 i 为个体排序序号,$1 \leqslant \eta^+ \leqslant 2$, $\eta^- = 2 - \eta^+$

Michalewicz 提出了线性排序的选择概率计算公式:

$$p_i = c(1-c)^{i-1} \tag{2.51}$$

上式中 i 为个体排序序号，c 为排序第 1 的个体的选择概率。

下面介绍几个常用的选择方法：

(1) **轮盘赌选择法**(roulette wheel selection) 这是最简单的一种选择方法。表 2.3 表示了 11 个个体适应度、选择概率和累积概率。为了选择交配个体，需要进行多轮选择。每一轮产生一个[0,1]均匀随机数，将该随机数作为选择指针来确定被选个体。如图 2.4 所示，第 1 轮随机数为 0.81，则第 6 个个体被选中，第 2 轮随机数为 0.32，则第 2 个个体被选中；依此类推，第 3,4,5,6 轮随机数为 0.96,0.01,0.65,0.42，则第 9,1,5,3 个个体依次被选中。这样经过选择产生的交配种群由以下个体组成：1,2,3,5,6,9。

表 2.3 轮盘赌选择法的选择概率计算

个体	1	2	3	4	5	6	7	8	9	10	11
适应度	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.1
选择概率	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.0
累积概率	0.18	0.34	0.49	0.62	0.73	0.82	0.89	0.95	0.98	1.00	1.00

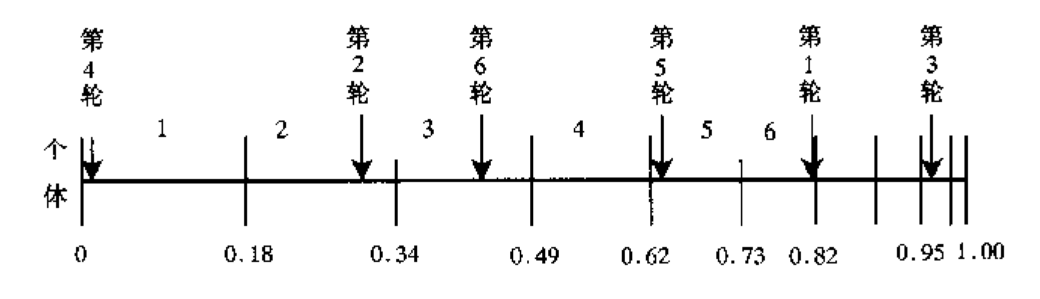


图 2.4 轮盘赌选择法

(2) **随机遍历抽样法**(stochastic universal sampling) 随机遍历抽样法提供了零偏差和最小个体扩展。设定 *npointer* 为需要选择的个体数目，等距离选择个体，选择指针的距离为 1/*npointer*，第一个指针的位置由[0, 1/*npointer*]区间的均匀随机数决定。

如图 2.5 所示，需要选择 6 个个体，指针间的距离为 1/6＝0.167，第一个指针的随机位置为 0.1，按这种选择方法被选中作为交配集个体为：1,2,3,4,6,8。

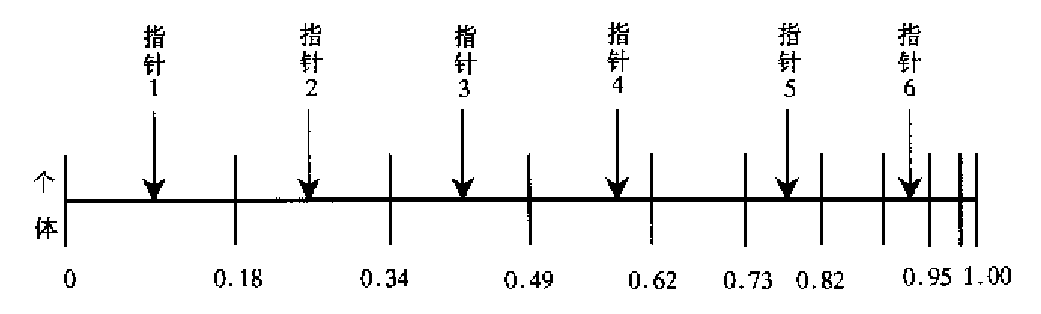


图 2.5 随机遍历抽样法

(3) **局部选择法**(local selection)　在局部选择法中,每个个体处于一个约束环境中,称为局部邻集(而其他选择方法中视整个种群为个体之邻集),个体仅与其邻近个体产生交互,该邻集的定义由种群的分布结构给出,邻集可被当作潜在的交配伙伴。

首先均匀随机地选择一半交配种群,选择方法可以是随机遍历方法也可以是截断选择方法,然后对每个被选个体定义其局部邻集,在邻集内部选择交配伙伴。邻集的结构可以是以下三种:

(1) **线形邻集**　整环型和半环型,如图2.6所示。

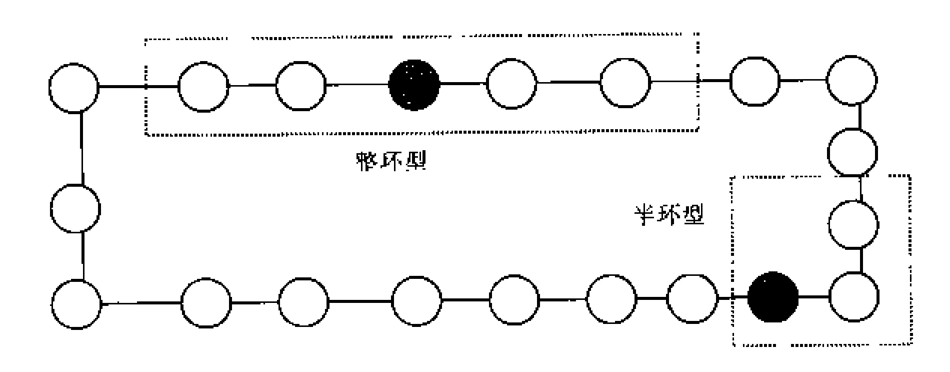


图2.6　线形邻集(距离为2)

(2) **两对角邻集**　整对角型和半对角型、整星型和半星型,如图2.7所示。

表2.4给出了给定结构和不同距离的邻集个体大小。

表2.4　给定结构和不同距离的邻集个体大小

结　构	距　离	
	1	2
整环型	2	4
半环型	1	2
整对角型	4	8
半对角型	2	4
整星型	8	24
半星型	3	8

种群中的个体之间存在距离,邻集越小,隔离距离越大。但是由于邻集间有重叠,会产生新的变体,这也就保证了所有个体之间的信息交换。邻集的大小决定了这种信息传播的速度,因此也必须在快速传播速度与保持种群多样化之间做出选择。通常求得较好的多样化,可以防止过早收敛到局部最小。对局部选择而言,在一个小邻集里进行优越于在一个大邻集里进行。但在种群范围内提供相互连接关系是必须的。我们推荐采用二维邻集、半星型结构、距离为1作为局部选择的方案。如果种群大小较大(超过100),可选用较大的距离或其他的二维邻集。

(3) **截断选择法**(truncation selection)　与前面几种自然方式的选择方法比较,截断选择法是一种人工选择方法。它适合于大种群。在截断选择法中,个体按适应度排序。只有最优

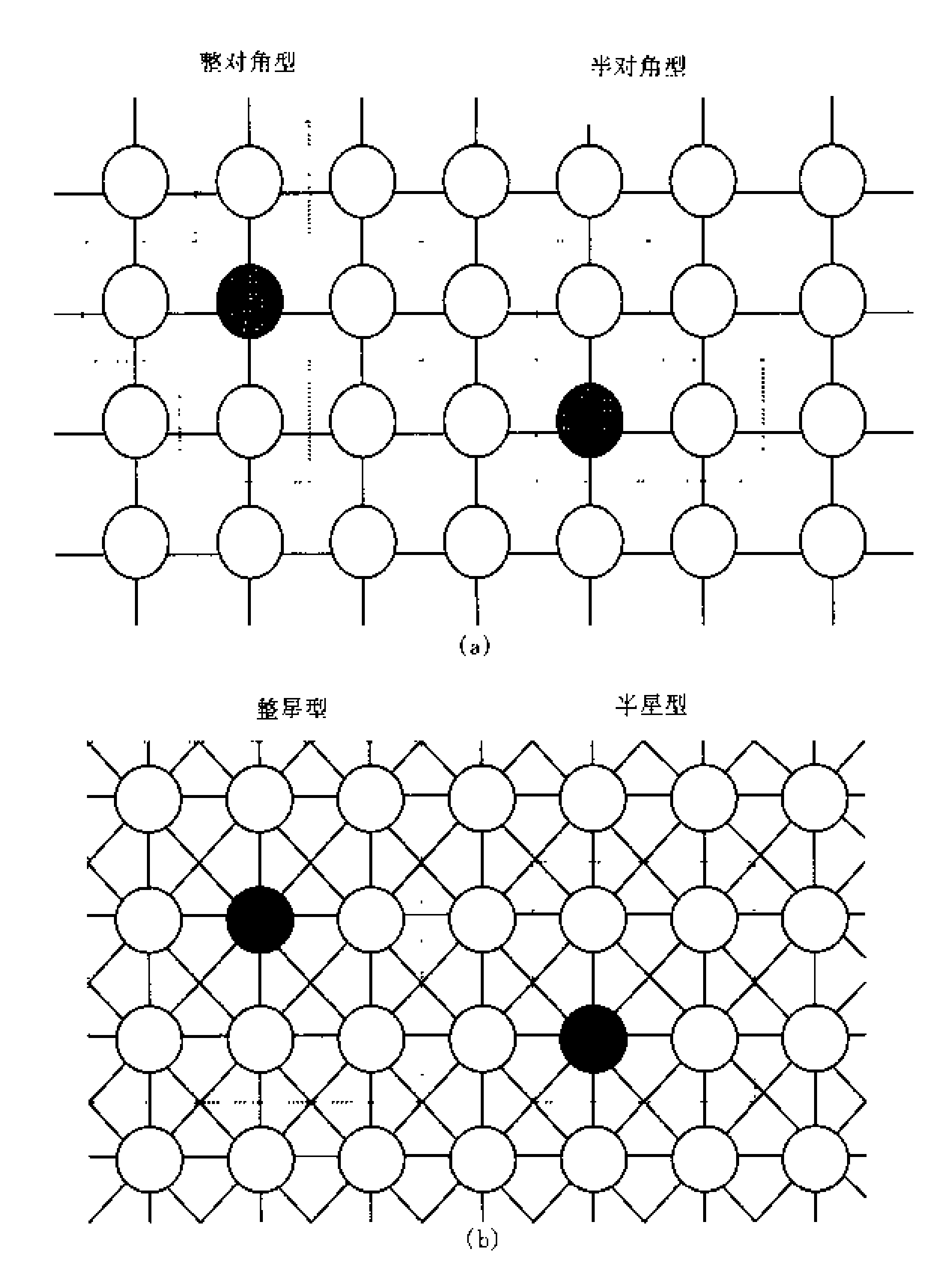


图 2.7 两对角邻集(距离为 1)
(a)整对角形和半对角型； (b)整星型和半星型

秀的个体能够被选作父个体，截断选择的参数叫做截断阈值 *Trunc*。它定义为被选做父个体的百分比，取值范围为 50%～10%。在该阈值之下的个体不能产生子个体。通常选择强度与截断阈值的关系见表 2.5 所示。

表 2.5 选择强度与截断阈值的关系

截断阈值	1%	10%	20%	40%	50%	80%
选择强度	2.66	1.76	1.2	0.97	0.8	0.34

选择强度：
$$SelInt_{Trunc}(Trunc)=\frac{1}{Trunc\sqrt{2\pi}e^{-f_c^2/2}} \tag{2.52}$$

多样化损失：
$$LossDiv_{Trunc}(Trunc)=1-Trunc \tag{2.53}$$

选择方差：$SelVar_{Trunc}(Trunc)=1-SelInt_{Trunc}(Trunc)(SelInt_{Trunc}(Trunc)-f_c)$ (2.54)

上式中 f_c 为下列高斯分布的积分下限：

$$Trunc=\int_{f_c}^{-\infty}\frac{1}{\sqrt{2\pi}}e^{-\frac{f^2}{2}}df \tag{2.55}$$

(4) **锦标赛选择法**(tournament selection) 在锦标赛选择法中，随机地从种群中挑选一定数目($Tour$)个体，然后将最好的个体选做父个体。这个过程重复进行完成个体的选择。锦标赛选择的参数为竞赛规模 $Tour$，其取值范围为[2, $Nind$]。表 2.6 表示了竞赛规模与选择强度之间的关系。

表 2.6 竞赛规模与选择强度之间的关系

竞赛规模	1	2	3	5	10	30
选择强度	0	0.56	0.85	1.15	1.53	2.04

选择强度： $$SelInt_{Tour}(Tour)=\sqrt{2(\log(Tour)-\log\sqrt{4.14\log(Tour)})} \tag{2.56}$$

多样化损失： $$LossDiv_{Tour}(Tour)=Tour^{-\frac{1}{Tour-1}}-Tour^{-\frac{Tour}{Tour-1}} \tag{2.57}$$

当 $Tour=5$ 时多样化损失大约为 50%。

选择方差： $$SelVar_{Tour}(Tour)=1-0.096\log(1+7.11(Tour-1)) \tag{2.58}$$

$$SelVar_{Tour}(2)=1-\frac{1}{\pi} \tag{2.59}$$

上述几种选择方法均以适应度为基础进行选择，这就可能在进化过程中导致以下的问题：

① 在种群中出现个别或极少数适应度相当高的个体时，采用这样的选择机制就有可能导致这些个体在种群中迅速繁殖。经过少数几次迭代后占满了种群的位置。这样，遗传算法的求解过程就结束了，也即收敛了。但这样很有可能是收敛到局部最优解，即遗传算法的不成熟收敛，即早熟现象的出现，这是因为搜索的范围很有限。因此一般不希望有个别个体在遗传算法运算的最初几次迭代时就在种群中占据主导地位。

② 当种群中个体适应度彼此非常接近时，这些个体进入配对集的机会相当，而且交配后得到的新个体也不会有多大变化。这样，搜索过程就不能有效地进行，选择机制有可能趋向于纯粹的随机选择，从而是进化过程陷于停顿的状态，难以找到全局最优解。

针对上述问题，可以采用适应度函数尺度变换(本章 2.3 节介绍)的方法来解决。另外，还可以采用以下的几种提高遗传算法性能的选择方法：

(1) **稳态繁殖**(steady state reproduction) 在迭代过程中用部分优质新子个体来更新种群中部分父个体来作为下一代种群。

(2) **没有重串的稳态繁殖**(steady state reproduction without duplicates) 在形成新一代种群时，使其中的种群均不重复。做法是：在将某个个体加入到新的一代种群之前，先检查该个体与种群中现有的个体是否重复，如果重复就舍弃。这种做法会明显改善遗传算法的行为，因为其增大了个体在种群中的分布区域，但增加了计算时间。

不同选择方法的行为是有差别的，基本遗传算法达到收敛的世代数与选择强度成反比，较高的选择强度是很好的选择方法，但太高会导致收敛过快，解的质量差。最小限度的种群大小往往依赖于目标函数的维数和选择强度，而选择强度又与选择参数(如选择压力、截断阀值、竞

赛规模)有关。锦标赛选择法只能赋离散值,线性排序选择法只允许较小区间值的选择强度。截断选择会导致比排序选择和锦标赛选择更高的多样化损失,截断选择倾向于用更好的个体取代较差的个体,因为所有低于适应度阀值之下的个体没有机会被选择,排序选择与锦标赛选择比较相似,但是排序选择往往用在锦标赛选择法因其离散性不能发挥作用的场合。对于同样的选择强度,截断选择的选择方差比排序选择和锦标赛选择小。

2.4.2 交叉/基因重组(crossover/recombination)

基因重组是把两个父个体的部分结构加以替换重组而生成新个体的操作,也称交叉(crossover)。重组的目的是为了能够在下一代产生新的个体,就像人类社会的婚姻过程,通过重组交叉操作,遗传算法的搜索能力得以飞跃地提高。基因重组和交叉是遗传算法获取新优良个体的最重要的手段。

(1) 实值重组

① 离散重组

离散重组在个体之间交换变量的值,考虑如下含有三个变量的个体:

父个体 1	12	25	5
父个体 2	123	4	34

子个体的每个变量可按等概率随机地挑选父个体,例如重组之后一个子个体为:

子个体 1	123	4	5
子个体 2	12	4	5

图 2.8 表示了离散重组后子个体的可能位置。

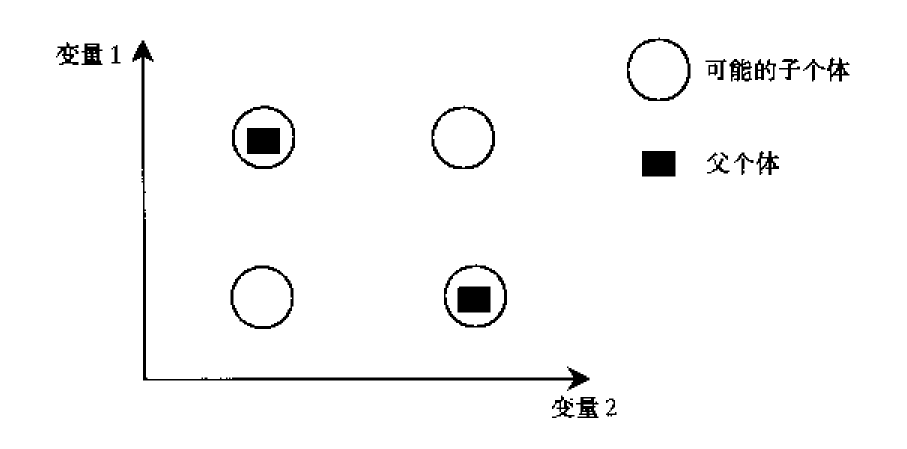


图 2.8 离散重组

② 中间重组

中间重组只能适用于实变量,而非二进制变量,见图 2.9。

子个体的产生按下列公式:

$$子个体 = 父个体1 + \alpha * (父个体2 - 父个体1) \tag{2.60}$$

这里 α 是一个比例因子,可由 $[-d, 1+d]$ 上均匀分布随机数产生。对于中间重组 $d=0$;一般选择 $d=0.25$。子代的每个变量的值按上面的表达式计算,对每个变量要选择一个新的 α 值。图 2.10 为按父个体变量值定义的子个体取值范围。考虑含有三个变量的两个个体:

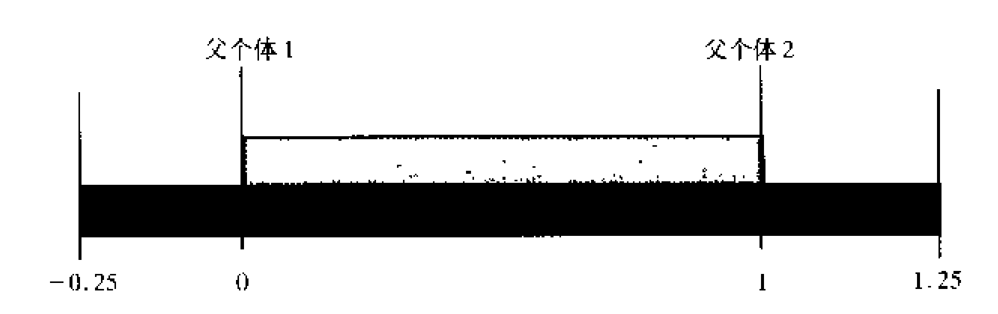


图 2.9　中间重组

父个体 1	12	25	5
父个体 2	123	4	34

α 值的样本为：

样本 1	0.5	1.1	−0.1
样本 2	0.1	0.8	0.5

计算出新的子个体为：

子个体 1	67.5	1.9	2.1
子个体 2	23.1	8.2	19.5

图 2.10 为中间重组后子个体的可能位置。

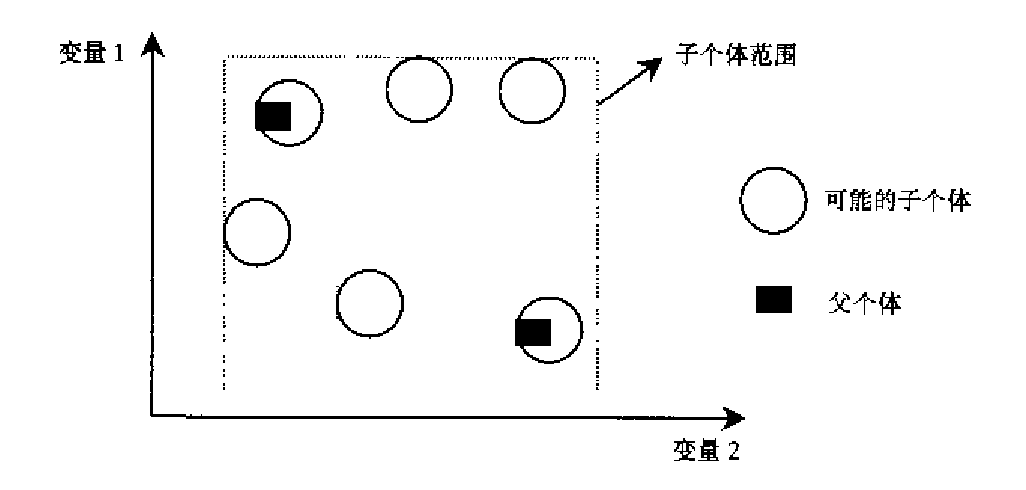


图 2.10　按父个体变量值定义的子个体取值范围

③ 线性重组

线性重组与中间重组比较相似，只是对所有变量只有一个 α 值。

父个体 1	12	25	5
父个体 2	123	4	34

α 值的样本为：

样本 1	0.5
样本 2	0.1

计算出新的子个体为：

子个体 1	67.5	14.5	19.5
子个体 2	23.1	22.9	7.9

图 2.11 为线性重组后子个体的可能位置。

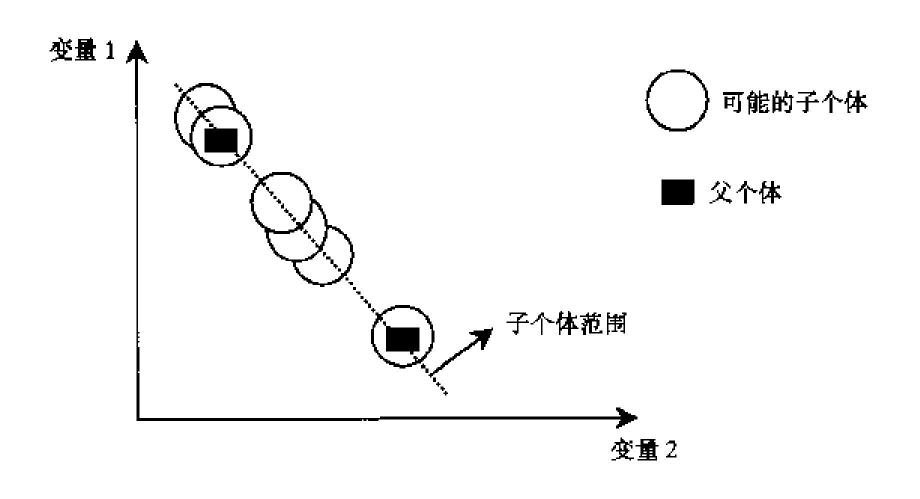


图 2.11 线性重组

(2) **二进制交叉**

① 单点交叉

单点交叉中，交叉点 k 的范围为[1, $Nvar-1$], $Nvar$ 为个体变量数目，在该点为分界相互交换变量。

考虑如下两个 11 位变量的父个体：

父个体 1	0	1	1	1	0	0	1	1	0	1	0
父个体 2	1	0	1	0	1	1	0	0	1	0	1

交叉点的位置为 5，如图 2.12 所示，交叉后生成两个子个体：

子个体 1	0	1	1	1	0	1	0	0	1	0	1
子个体 2	1	0	1	0	1	0	1	1	0	1	0

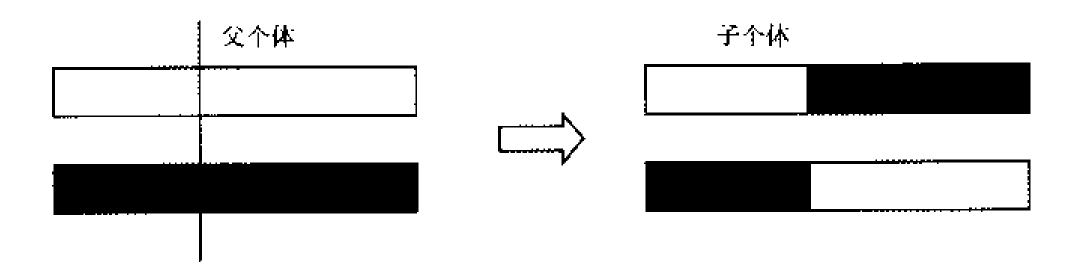


图 2.12 单点交叉

② 多点交叉

对于多点交叉，m 个交叉位置 K_i 可无重复随机地选择，在交叉点之间的变量间续地相互交换，产生两个新的后代，但在第一位变量与第一个交叉点之间的一段不做交换。

考虑如下两个 11 位变量的个体：

父个体 1　　0 1 1 1 0 0 1 1 0 1 0

父个体 2　　1 0 1 0 1 1 0 0 1 0 1

交叉点的位置为：　　2　6　10

如图 2.13 所示，交叉后两个新个体为：

子个体 1　　0 1 1 0 1 1 1 1 0 1 1

子个体 2　　1 0 1 1 0 0 0 0 1 0 0

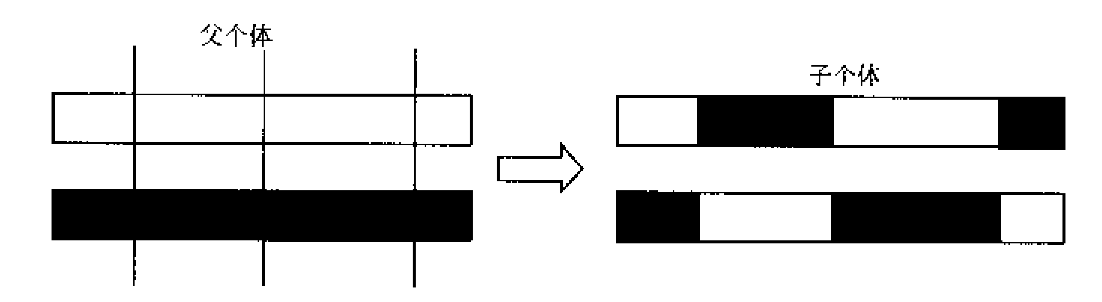


图 2.13　多点交叉

多点交叉的思想源于控制个体特定行为的染色体表示信息的部分无须包含于邻近的子串中，多点交叉的破坏性可以促进解空间的搜索，而不是促进过早地收敛。因此搜索更加健壮。

③ 均匀交叉

单点和多点交叉的定义使得个体在交叉点处分成片段。均匀交叉更加广义化，将每个点都作为潜在的交叉点。随机地产生与个体等长的 0－1 掩码，掩码中的片段表明了哪个父个体向子个体提供变量值。

考虑如下两个 11 位变量的个体：

父个体 1　　0 1 1 1 0 0 1 1 0 1 0

父个体 2　　1 0 1 0 1 1 0 0 1 0 1

掩码样本（1 表示父个体 1 提供变量值，0 表示父个体 2 提供变量值）：

样本 1　　0 1 1 0 0 0 1 1 0 1 0

样本 2　　1 0 0 1 1 1 0 0 1 0 1

交叉后两个新个体为：

子个体 1　　1 1 1 0 1 1 1 1 1 1 1

子个体 2　　0 0 1 1 0 0 0 0 0 0 0

均匀交叉类似于多点交叉，可以减少二进制编码长度与给定参数特殊编码之间的偏差。它的算法与离散重组是等价的。

除了上述交叉方法以外，还有部分匹配交叉（Partially Matched Crossover, PMX）、顺序交叉（Ordered Crossover, OX）、循环交叉（Cycle Crossover, CX）、洗牌交叉（shuffle crossover）、缩小代理交叉（crossover with reduced surrogate）等，前三种交叉方法的描述见 7.1 节。

2.4.3　变异（mutation）

重组之后是子代的变异，子个体变量以很小的概率或步长产生转变，变量转变的概率或步长与维数（即变量的个数）成反比，与种群的大小无关。据研究，对于单峰函数 $1/n$ 是最好的选择，开始时增加变异率，结束时减少变异率可以改善搜索速度。但对于多峰函数变异率的自

适应过程是很有益的选择。变异本身是一种局部随机搜索，与选择/重组算子结合在一起，保证了遗传算法的有效性，使遗传算法具有局部的随机搜索能力；同时使得遗传算法保持种群的多样性，以防止出现非成熟收敛。在变异操作中，变异率不能取得太大，如果变异率大于 0.5，遗传算法就退化为随机搜索，而遗传算法的一些重要的数学特性和搜索能力也不复存在了。

（1）**实值变异**

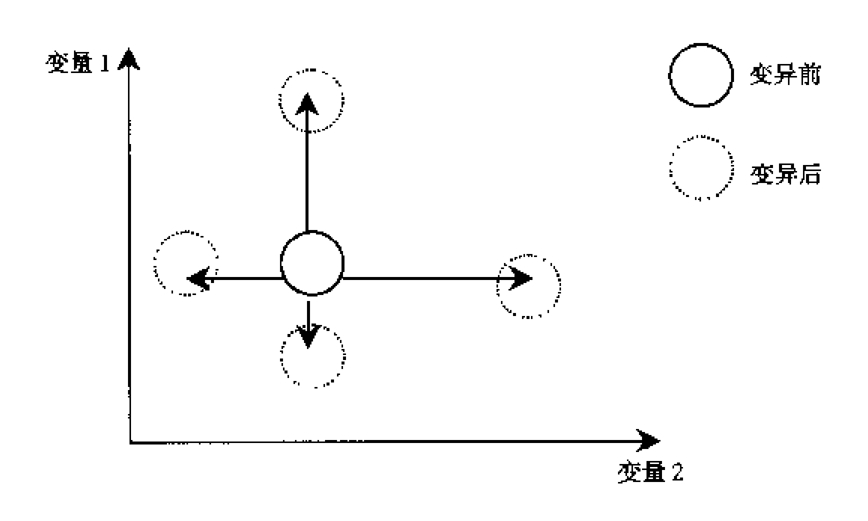


图 2.14 实值变异

变异步长的选择比较困难，最优的步长视具体情况而定，甚至在优化过程中可以改变。通常较小的步长会比较成功，但有时大步长比较快些。建议采用如下的变异算子：

$$X' = X \pm 0.5L\Delta \tag{2.61}$$

上式中，$\Delta = \sum_{i=0}^{m-1} \frac{a(i)}{2^i}$，$a(i)$以概率 $1/m$ 取值 1，以 $1-1/m$ 取值 0，通常 $m=20$；L 为变量的取值范围；X 为变异前变量取值；X'为变异后变量取值。

（2）**二进制变异**

对于二进制编码的个体而言，变异意味着变量的翻转。对于每个个体，变量值的改变是随机的，如下所示，有 11 位变量的个体，第 4 位发生了翻转。

变异前： 01110011010

变异后： 01100011010

上述个体解码成实数，变异的效果依赖于实际的编码方法。除了上述基本位变异法以外，还有其他的变异方法，如换位（displacement）、复制（duplication）、插入（addition）、删除（deletion）等等。

2.5 算法设计与实现

基本遗传算法（Simple Genetic Algorithm, SGA）可定义为一个 8 元组：

$$SGA = (C, E, P_0, M, \Phi, \Gamma, \Psi, T) \tag{2.62}$$

式中， C —— 个体的编码方法，SGA 使用固定长度二进制符号串的编码方法；

E —— 个体的适应度评价函数；

P_0—— 初始群体；

M —— 群体大小，一般取 20～100；

Φ —— 选择算子，SGA 使用比例选择算子；

Γ —— 交叉算子，SGA 使用单点交叉算子；

Ψ —— 变异算子，SGA 使用基本位变异算子；

T —— 算法终止条件，一般终止进化代数为 100～500。

2.5.1 问题的表示

对于一个实际的待优化问题，首先需要将其表示为适合于遗传算法操作的形式。这里以二进制编码为例，它包括以下几个步骤：

① 根据具体问题确定待寻优的参数。

② 对每个参数确定它的变化范围，并用一个二进制数来表示。例如，若参数 a 的变化范围为$[a_{min}, a_{max}]$，用 m 位二进制数 b 来表示，则二者之间满足：

$$a = a_{min} + \frac{b}{2^m - 1}(a_{max} - a_{min}) \tag{2.63}$$

这时参数范围的确定应覆盖全部的寻优空间，字长 m 的确定应在满足精度要求情况下尽量取小的 m，以尽量减小遗传算法计算的复杂性。

③ 将所有表示参数的二进制数串接起来组成一个长的二进制字串。该字串的每一位只有 0 或 1 两种取值。该字串即为遗传算法的操作对象。

2.5.2 初始种群的产生

产生初始种群的方法通常有两种。一种是完全随机的方法产生的，它适合于对问题的解无任何先验知识的情况。某些先验知识可转变为必须满足的一组要求，然后在满足这些要求的解中再随机地选取样本。这样选择初始种群可使遗传算法更快地到达最优解。

2.5.3 算法设计与实现

附录Ⅱ.1 给出了基本遗传算法的 C 语言源程序，它是根据 David, E. Goldberg 的 Pascal 源程序改写的，用于 2.1 节的函数优化问题。下面我们讨论基本遗传算法实现方法的几个主要问题。

1. 数据结构与遗传算法参数

基本遗传算法处理的对象主要是个体，因此设计了结构变量 individual 来描述个体信息，其中包括个体的染色体串 chrom，个体适应度 fitness，个体对应的变量 varible，交叉位置 xsite，以及记录父个体编号 parent [2]等。为记录进化历代最佳个体，设计结构变量 bestever，表示最佳个体对应的染色体 chrom，个体适应度 fitness，对应的变量 varible 以及最佳个体产生的代数。根据个体变量定义，设计当前代种群 oldpop 以及新一代种群 newpop 为全局变量。此外，将当前种群中个体最大适应度 max、个体平均适应度 avg、最小适应度 min、个体适应度累计值也定义为全局变量。

基本遗传算法运行参数包括：种群大小 popsize，染色体长度 lchrom，进化最大代数 maxgen，总运行次数 maxruns，交叉率 pcross，变异率 pmutation 等。这些参数在运行开始时由使用者输入，参见函数 initdata()。

```
/* 全局变量 */
struct individual                        /* 个体 */
{
    unsigned    *chrom;                  /* 染色体 */
    double      fitness;                 /* 个体适应度 */
    double      varible;                 /* 个体对应的变量值 */
    int         xsite;                   /* 交叉位置 */
    int         parent[2];               /* 父个体 */
    int         *utility;                /* 特定数据指针变量 */
};
struct bestever                          /* 最佳个体 */
{
    unsigned    *chrom;                  /* 最佳个体染色体 */
    double      fitness;                 /* 最佳个体适应度 */
    double      varible;                 /* 最佳个体对应的变量值 */
    int         generation;              /* 最佳个体生成代 */
};
struct individual  *oldpop;              /* 当前代种群 */
struct individual  *newpop;              /* 新一代种群 */
struct bestever bestfit;                 /* 最佳个体 */
double sumfitness;                       /* 种群中个体适应度累计 */
double max;                              /* 种群中个体最大适应度 */
double avg;                              /* 种群中个体平均适应度 */
double min;                              /* 种群中个体最小适应度 */
float  pcross;                           /* 交叉率 */
float  pmutation;                        /* 变异率 */
int    popsize;                          /* 种群大小 */
int    lchrom;                           /* 染色体长度 */
int    chromsize;                        /* 存储一染色体所需字节数 */
int    gen;                              /* 当前世代 */
int    maxgen;                           /* 最大世代数 */
int    run;                              /* 当前运行次数 */
int    maxruns;                          /* 总运行次数 */
```

2. 产生初始种群

为产生初始种群设计的函数为 initpop()。种群中个体的染色体编码的每一位按等概率在 0 与 1 中选择。在产生染色体编码后，对个体进行解码和适应度计算。该算法程序如下：

```
void initpop()   /* 随机初始化种群 */
{
    int j, j1, k, stop;
    unsigned mask = 1;
```

```
    for(j=0; j < popsize; j++)
    {
        for(k=0; k < chromsize; k++)
        {
            oldpop[j].chrom[k]=0;
            if(k==(chromsize-1))
                stop=lchrom -(k*(8*sizeof(unsigned)));
            else
                stop=8*sizeof(unsigned);
            for(j1=1; j1 <=stop; j1++)
            {
                oldpop[j].chrom[k]=oldpop[j].chrom[k]<<1;
                if(flip(0.5))
                    oldpop[j].chrom[k]=oldpop[j].chrom[k]|mask;
            }
        }
            oldpop[j].parent[0]=0;            /* 初始父个体信息 */
            oldpop[j].parent[1]=0;
            oldpop[j].xsite=0;
        objfunc(&(oldpop[j]));            /* 计算初始适应度 */
    }
}
```

解码和适应度计算用 objfunc()函数。这里目标函数用 2.1 式的函数 $f(x)=x\sin(10\pi x)+2$，优化变量区间为$[-1,2]$。

个体的适应度 *critter.fitness* 计算如下：

$$critter.fitness = critter.varible * \sin(10\pi * critter.varible) + 2$$

上述算法的程序如下：

```
void objfunc(critter)            /* 计算适应度函数值 */
struct individual *critter;
{
    unsigned mask=1;
    unsigned bitpos;
    unsigned tp;
    double pow(), bitpow ;
    int j, k, stop;
    critter->varible=0.0;
    for(k=0; k < chromsize; k++)
    {
        if(k==(chromsize-1))
            stop=lchrom-(k*(8*sizeof(unsigned)));
```

```
        else
            stop = 8 * sizeof(unsigned);
        tp = critter->chrom[k];
        for(j = 0; j < stop; j++)
        {
            bitpos = j + (8 * sizeof(unsigned)) * k;
            if((tp&mask) == 1)
            {
                bitpow = pow(2.0, (double)bitpos);
                critter->varible = critter->varible + bitpow;
            }
            tp = tp>>1;
        }
    }
    critter->varible = -1 + critter->varible * 3/(pow(2.0, (double)lchrom) - 1);
    critter->fitness = critter->varible * sin(critter->varible * 10 * atan(1) * 4) + 2.0;
}
```

3. 遗传操作设计

为轮盘赌选择设计的函数 select(),返回种群中被选择的个体编号。方法是产生一个[0,1]随机数 $pick$,若 $pick < sum = \sum_{i=0}^{i} oldpop[i]/sumfitness$,则第 i 个个体被选中。该算法程序如下:

```
int select()   /* 轮盘赌选择 */
{
    extern float randomperc();
    float sum, pick;
    int i;
    pick = randomperc();
    sum = 0;
    if(sumfitness != 0)
    {
        for(i = 0; (sum < pick)&&(i < popsize); i++)
            sum += oldpop[i].fitness/sumfitness;
    }
    else
        i = rnd(1, popsize);
    return(i - 1);
}
```

为单点交叉操作设计的函数 crossover(),由父个体 parent1 和 parent2 产生子个体 child1

和 child2，若交叉发生处理编码赋值，并返回交叉点位置 jcross；否则不做任何处理，返回 0。方法是，先通过 flip(pcross)函数确定是否发生交叉操作，若发生交叉操作，在[1，lchrom]区间随机确定交叉位置 jcross，child1 继承 parent1 在 jcross 之前的编码和 parent2 在 jcross 之后的编码，child2 继承 parent2 在 jcross 之前的编码和 parent1 在 jcross 之后的编码。编码赋值按染色体的编码位逐一判断处理。该算法程序如下：

```
int crossover(unsigned *parent1, unsigned *parent2, unsigned *child1, unsigned *child2)
/* 由两个父个体交叉产生两个子个体 */
{
    int j, jcross, k;
    unsigned mask, temp;
    if(flip(pcross))
    {
        jcross=rnd(1,(lchrom - 1));        /* 交叉位置在 1 和 l-1 之间 */
        ncross++;
        for(k=1; k<=chromsize; k++)
        {
            if(jcross>=(k*(8*sizeof(unsigned))))
            {
                child1[k-1]=parent1[k-1];
                child2[k-1]=parent2[k-1];
            }
            else if((jcross<(k*(8*sizeof(unsigned))))&&
                (jcross>((k-1)*(8*sizeof(unsigned)))))
            {
                mask=1;
                for(j=1; j<=(jcross-1-((k-1)*(8*sizeof(unsigned)))); j++)
                {
                    temp=1;
                    mask=mask<<1;
                    mask=mask|temp;
                }
                child1[k-1]=(parent1[k-1]&mask)|(parent2[k-1]&(~mask));
                child2[k-1]=(parent1[k-1]&(~mask))|(parent2[k-1]&mask);
            }
            else
            {
                child1[k-1]=parent2[k-1];
                child2[k-1]=parent1[k-1];
            }
        }
    }
```

```
    else
    {
        for(k=0; k < chromsize; k++)
        {
            child1[k]=parent1[k];
            child2[k]=parent2[k];
        }
        jcross=0;
    }
    return(jcross);
}
```

为变异操作设计的函数 mutation(),按变异概率 pmutation 确定个体 child 的编码位是否发生操作,若某编码位发生变异,则该编码翻转。该算法程序如下:

```
void mutation(unsigned *child)  /* 变异操作 */
{
    int j, k, stop;
    unsigned mask, temp=1;
    for(k=0; k < chromsize; k++)
    {
        mask=0;
        if(k==(chromsize-1))
            stop=lchrom -(k*(8*sizeof(unsigned)));
        else
            stop=8*sizeof(unsigned);
        for(j=0; j < stop; j++)
        {
            if(flip(pmutation))
            {
                mask=mask|(temp<<j);
                nmutation++;
            }
        }
        child[k]=child[k]^mask;
    }
}
```

4. 世代进化过程的实现

为模拟世代进化过程设计的函数 generation(),以种群的处理对象实现了一个世代内的三种遗传操作。首先在当前种群中用 select()函数选择两个个体,然后对两个个体按交叉概率实行可能的交叉操作和变异操作,然后对个体解码、计算适应度、记录亲子信息数据等,生成新一

代个体。该算法程序如下:

```
void generation()
{
  int mate1, mate2, jcross, j=0;
  preselect();
  /* 选择,交叉,变异 */
  do
    {
      /* 挑选交叉配对 */
      mate1=select();
      mate2=select();
      /* 交叉和变异 */
      jcross=crossover(oldpop[mate1].chrom, oldpop[mate2].chrom,
            newpop[j].chrom, newpop[j+1].chrom);
      mutation(newpop[j].chrom);
      mutation(newpop[j+1].chrom);
      /* 解码,计算适应度 */
      objfunc(&(newpop[j]));
      /* 记录亲子关系和交叉位置 */
      newpop[j].parent[0]=mate1+1;
      newpop[j].xsite=jcross;
      newpop[j].parent[1]=mate2+1;
      objfunc(&(newpop[j+1]));
      newpop[j+1].parent[0]=mate1+1;
      newpop[j+1].xsite=jcross;
      newpop[j+1].parent[1]=mate2+1;
      j=j + 2;
    }
  while(j <(popsize-1));
}
```

5. 主程序

主程序设计了多次运行遗传算法的过程,每次运行前由使用者输入不同的参数设置,运行结束时将有关进化过程的统计结果写入输出数据文件中。作为一次的遗传算法运行,首先进行初始化 initilaize(),初始化的工作包括获取算法参数,计算染色体字节长度 chromsize,分配数据空间,初始化随机数发生器以及产生初始种群,并输出初始代统计信息等。然后进入共maxgen 代的进化计算。每一代进化计算调用generation()函数产生新一代种群,调用statistics()计算新一代种群的统计信息,并将这些统计信息写入输出数据文件。每次运行结束后调用函数 freeall()释放内存空间。主程序如下:

```
main(argc, argv)     /* 主程序 */
```

```
int argc;
char *argv[];
{
    struct individual *temp;
    FILE  *fopen();
    void  title();
    char  *malloc();
        if((outfp=fopen(argv[1],"w"))==NULL)
        {
            fprintf(stderr,"Cannot open output file %s\n",argv[1]);
            exit(-1);
        }
    g_init();
    setcolor(9);
    setbkcolor(15);
    title();
    disp_hz16("输入遗传算法执行次数(1-5):",100,120,20);
    gscanf(320,120,9,15,4,"%d",&maxruns);
    for(run=1; run<=maxruns; run++)
    {
      initialize();
      for(gen=0; gen<maxgen; gen++)
        {
        fprintf(outfp,"\n第 %d / %d 次运行: 当前代为 %d,共 %d 代\n",
              run,maxruns,gen,maxgen);
            /* 产生新一代 */
        generation();
            /* 计算新一代种群的适应度统计数据 */
        statistics(newpop);
            /* 输出新一代统计数据 */
        report();
        temp=oldpop;
        oldpop=newpop;
        newpop=temp;
        }
        freeall();
        }
    }
```

在运行遗传算法程序时,需要对一些参数作事先选择,它们包括种群的大小、染色体长度、交叉率、变异率、最大进化代数等,这些参数对遗传算法的性能都有很重要的影响。一般说来,选择较大数目的初始种群可以同时处理更多的解,因而容易找到全局最优解,其缺点是增加了

每次迭代的时间，一般取 20～100。交叉率的选择决定了交叉操作的频率，频率越高，可以越快地收敛到最有希望的最优解区域，因此一般选取较大的交叉率，但太高的频率也可能导致过早收敛，一般取值 0.4～0.9。变异率的选取一般受种群大小、染色体长度等因素影响，通常选取很小的值，一般取 0.001～0.1。若选取高的变异率，虽然增加样本模式的多样性，但可能会引起不稳定。种群大小及染色体长度越大，变异率选取越小。染色体长度主要决定于问题求解精度的要求，精度越高，染色体长度越长，搜索空间越大，相应地要求种群大小设置大一些。最大进化代数作为一种模拟终止条件，一般视具体问题而定，取 100～500 代。对于具体问题而言，衡量参数设置恰当与否，要依据多次运行的收敛情况和解的质量来判断。如果调整参数难以有效地提高遗传算法的性能，则往往需要借助对基本遗传算法的改进，改进的手段可以是多方面的，如适应度比例调整、引入自适应交叉率和变异率，尝试其他的遗传操作（与基本遗传算法不同的选择、交叉和变异法），也可以采用一些混合方法等（详见第 4 章内容）。

为了更好地把握基本遗传算法，下面给出 2.1 节优化实例的部分模拟实算统计结果。为简单起见，我们设置了如下的参数：

基本遗传算法参数

种群大小(popsize)	=30
染色体长度(lchrom)	=22
最大进化代数(maxgen)	=150
交叉率(pcross)	=0.80
变异率(pmutation)	=0.05

表 2.7、表 2.8 分别列出了初始代到第 1 代、第 7 代到第 8 代的种群进化统计结果。图 2.15显示了历代进化的个体适应度（最大值、最小值和平均值）变化曲线。

表 2.7　模拟计算统计报告(0～1 代)

个体	世代数 0 染色体编码	适应度	父个体	交叉位置	世代数 1 染色体编码	适应度
1	0111010100110001110010	2.041 164	(30,2)	0	1000110010000100100010	1.981 748
2	0000101010011011001001	1.450 069	(30,2)	0	0000101010011011001001	1.450 069
3	1111100011110011110100	2.443 676	(24,10)	6	1010001001010001001011	3.345 799
4	0001001001000010010101	1.445 781	(24,10)	6	1111010011101011100010	1.851 259
5	1111011110010110001011	3.422 826	(3,15)	20	1111100011110011110100	2.443 676
6	0000011011111110100111	2.254 731	(3,15)	20	1100001110001111111100	2.251 760
7	1100110000110110100101	1.090 242	(5,15)	5	1111001111101111111100	2.251 221
8	1110110100011001001001	1.665 942	(5,15)	5	1100011110010110001011	3.422 902
9	1111110111000010110010	2.009 216	(22,1)	1	0111010100110001110010	2.041 164
10	1111011001010011001001	1.494 669	(22,1)	1	0000011111011000111010	2.041 850
11	0000001000010111101011	0.857 773	(21,20)	15	0111101110011101111001	2.800 022
12	1010000011000001000010	2.164 800	(21,20)	15	0000000011001111101000	1.555 685
13	1101001100010000000110	1.906 060	(18,2)	10	0101110100010011000001	1.509 050
14	1011011010011011111011	2 864 249	(18,2)	10	0000101010111110011100	1.804 938
15	1100001111101111111100	2.251 226	(1,20)	8	0111010111001111111011	3.063 343

续表 2.7

个体	世代数 0 染色体编码	适应度	父个体	交叉位置	世代数 1 染色体编码	适应度
16	0111100001110000100010	2.003 401	(1,20)	8	0011000000110011110010	2.055 468
17	0010010010011000011110	2.137 394	(25,17)	2	0110010010011001011110	2.382 481
18	0101110100111110011100	1.804 015	(25,17)	2	0011001110110001101011	1.517 991
19	0001100011110011110111	2.200 482	(6,14)	19	0000011011111110110011	2.599 965
20	0000000011001111111011	3.058 461	(6,14)	19	1011111010011011111111	1.562 136
21	0111101110011111101000	1.579 198	(10,27)	9	1111011001010001100101	1.056 378
22	0000011111011000111010	2.041 850	(10,27)	9	0000011001010011001001	1.494 851
23	0101110111100100110010	1.993 267	(3,26)	13	1111100011110011000011	2.476 917
24	1010000011001011100010	1.852 255	(3,26)	13	0010001100111011110100	2.435 961
25	0011011110110001111010	1.986 588	(25,20)	21	0011011110110001111010	1.986 588
26	0010001100111011000011	2.382 383	(25,20)	21	0000000011001111010010	1.959 661
27	0000010001010001100001	1.561 055	(20,13)	13	0000000001001010000110	1.866 818
28	0100110000011000110010	1.983 764	(20,13)	13	1101001100011111111011	3.099 568
29	1111110010100000111110	2.450 350	(20,4)	6	0100001001000010010101	1.445 675
30	1000110010000100100010	1 981 748	(20,4)	6	0001000011001011011011	0.726796

第 1 代统计:

总交叉操作次数=14,总变异操作数=30

最小适应度:0.726 796　最大适应度:3.422 902　平均适应度 2.082 725

迄今发现最佳个体=>所在代数:1　适应度:3.422 902　染色体:1100011110010110001011

对应的变量值:1.456 889

表 2.8　模拟计算统计报告(8~9 代)

个体	世代数 7 染色体编码	适应度	父个体	交叉位置	世代数 8 染色体编码	适应度
1	1100111011011001111101	3.031 261	(10,7)	3	1010000101010011111011	2.742 680
2	1100000100110001110101	3.003 174	(10,7)	3	1001101100100000110101	2.504 004
3	1000101111100111111110	2.068 714	(27,14)	6	1100110101011000111110	2.443 574
4	1110101010011111111011	3.103 389	(27,14)	6	0000001111110001100101	1.053 324
5	0100001101101111001111	3.803 827	(27,30)	2	1100100010010001110001	2.586 133
6	0101000111010111111011	3.003 873	(27,30)	2	0000110111110001110100	2.443 441
7	1000000101010011111011	2.742 550	(7,21)	18	1000000101010011111100	2.247 591
8	0011111100100001110111	1.445 575	(7,21)	18	1111111111010101111101	3.093 265
9	1100111111101001110111	1.633 465	(26,3)	9	1011011001100111111110	2.072 600
10	1011111100100001110101	2.988 837	(26,3)	9	0000101111110110010111	0.359 490
11	1011001010010011111111	1.384 168	(17,7)	21	1010001011110011010100	2.103 103
12	1110001100110011010101	2.196 195	(17,7)	21	1000000101010011110011	3.091 589
13	0111111011010111111011	3.003 525	(21,18)	12	1111011111010111110111	2.495 166
14	0000000101011000111110	2.443 669	(21,18)	12	1111011110100101010100	1.986 367
15	1001111100100001110111	1.445 461	(22,16)	0	1001101010010101111011	2.455 440
16	0100111010010001111001	2.848 732	(22,16)	0	0100111010110001111001	2 847 207
17	1010001011110011010101	2.210 419	(30,5)	7	0000100101101111001111	3.804 290
18	1011011100100001110111	1.445 004	(30,5)	7	0100001010010000111001	2.668 579
19	0111111001011001111110	2.265 397	(14,21)	8	0000000111110101110100	2.451 121
20	0101101111010111011110	2.449 366	(14,21)	8	1111111101011000111110	2.443 431
21	1111111111010101110100	2.451 027	(11,24)	5	1011010011010001111101	2.961 478
22	1000101010010101111011	2.455 161	(11,24)	5	1101101010010011111111	1.384 760

续表 2.8

个体	世代数 7 染色体编码	适应度	父个体	交叉位置	世代数 8 染色体编码	适应度
23	1110101010011011110100	2.438 146	(20,6)	2	0100000111000111101011	0.886 470
24	1111110011010001111101	2.961 797	(20,6)	2	0100101101010101001111	3.808 676
25	0111111100110101111011	2.487 896	(4,16)	15	1100101010011111111001	2.641 112
26	1011011001110010110101	2.833 003	(4,16)	15	0100111010010001111011	2.165 550
27	1100111111110001110101	3.009 112	(30,1)	0	0000100110011011101000	1.474 243
28	1111101000011010111101	2.695 039	(30,1)	0	1100111011111001111101	3.047 101
29	0100101101010011111011	2.745 186	(4,19)	8	0110101001011001011110	2.384 070
30	0000100010010001111001	2.848 854	(4,19)	8	0011111010011111111011	3.104 397

第 8 代统计：

总交叉操作次数=94,总变异操作数=253

最小适应度:0.359 490　最大适应度:3.808 676　平均适应度 2.391 675

迄今发现最佳个体=＞所在代数:8　适应度:3.808 676　染色体:0100101101010101001111

对应的变量值:1.843 779

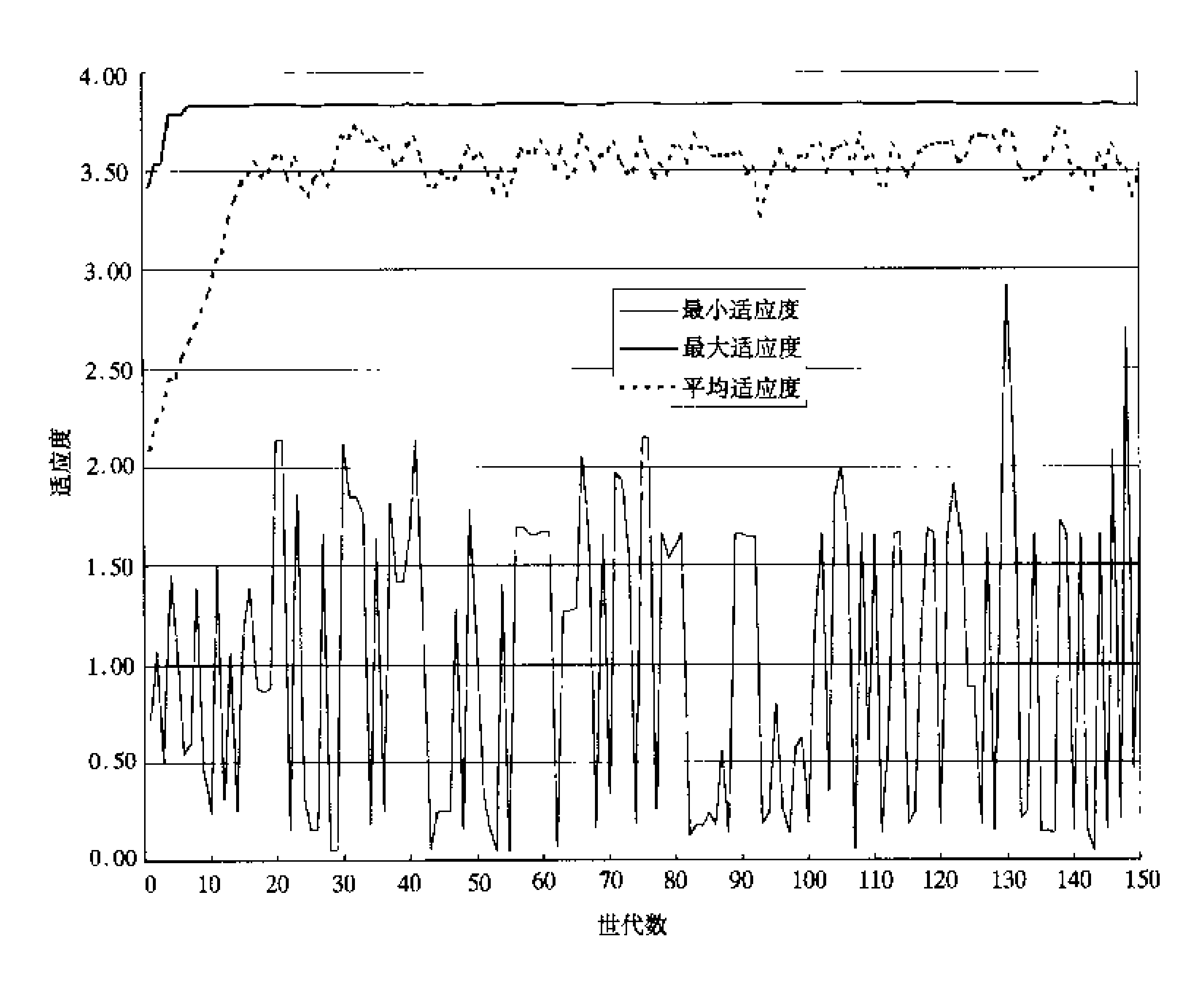


图 2.15　历代适应度进行曲线

参考文献

[1] Goldberg D E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA, Addison - Wisely, 1989

[2] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer - Verlag, Second, Extended Edition, 1994

[3] Blickle T, Thiele L. A Mathematical Analysis of Tournament Selection. In: Proceedings of the 6th International Conference(ICGA95), Morgan Kaufmann, San Francisco, CA, 1995

[4] Vose M D. Modeling Simple Genetic Algorithms. In: Foundations of Genetic Algorithms II, Morgan Kaufmann Publishers, 1993, 63~73

[5] Blickle T, Thiele L. A Comparison of Selection Schemes Used in the Genetic Algorithms. TIK - Report, Swiss Federal Institute of Technology(ETH), 1995

[6] Muhlenbein H. Evolutionary Algorithms: Theory and Applications. GMD Schloss Birlinhoven, 1995

[7] Syswerda G. Uniform Crossover in the Genetic Algorithms. In: Proceedings of the 3th International Conference(ICGA89), Morgan Kaufmann Publishers, 1989

[8] Whitely D. The GENITOR Algorithm and Selection Pressure: Why Rank - based Allocation of Reproductive Trials is Best. In: Proceedings of the Third International Conference(ICGA89), Morgan Kaufmann Publishers, 1989

[9] Goldberg D E, Deb K. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In: Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, 1989, 69~93

[10] Whitely D. Fundamental Principles of Deception in Genetic Search. In: Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, 1989, 221~241

[11] Muhlenbein H. Optimal Interaction of Mutation and Crossover in the Breeder Genetic Algorithm. GMD Schloss Birlinhoven, 1995

[12] Goldberg D E. The Existential Pleasures of Genetic Algorithms. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed). Wiley, 1995, 23~31

[13] Satyadas, Krishnakumar K. Genetic Algorithm Modules in MATLAB: Design and Implementation Using Software Engineering Practices. In: Genetic Algorithms in Engineering and Computer Science, Winter G (ed). Wiley, 1995, 321~344

[14] Annie S W. Non - Coding DNA and Floating Building Blocks for the Genetic Algorithm. Doctoral Dissertation, Department of Computer Science and Engineering, The University of Michigan, 1996

[15] Manderick, de Weger M. The Genetic Algorithm and the Structure of the Fitness Landscape, . In: Proceedings of the Fourth International Conference on Genetic Algorithms, CA, Morgan Kaufman, 1991, 143~150

[16] 章珂，刘贵忠．交叉位置非等概率选取的遗传算法．信息与控制，1997, 26, (1): 53~60

[17] 徐洪泽，陈桂林，张福恩．遗传算法单双点交叉方法的比较研究．哈尔滨工业大学学报，1999, 30 (2): 63~71

[18] 张晓缋，方浩等．遗传算法的编码机制研究．信息与控制，1997, 26(2): 134~139

[19] 张晓缋，戴冠中等．遗传算法种群多样性的分析研究．控制理论与应用，1998, 15(1): 17~22

[20] 辛香非，朱鳌鑫．遗传算法的适应度函数研究．系统工程与电子技术，1998(11): 58~62

[21] 王秀峰．实数编码的遗传算法及其在逆变器馈电交流电机中的应用．自动化学报，1998, 24(2): 250~253

第 3 章 遗传算法的数学基础

遗传算法在机理方面具有搜索过程和优化机制等属性，数学方面的性质可通过模式定理和构造块假设等分析加以讨论，Markov 链也是分析遗传算法的一个有效工具。就遗传算法的计算机理而言，在人工智能中光束搜索作为最佳优先搜索法，对搜索空间中计算量压缩有一定的效果，采用 open 表的数据结构，该表长度反映了光束的幅度，在搜索过程中不断地以新的更好的结点来调整该表。遗传算法的群体规模与光束搜索中的光束幅度类似。优化理论中的单纯形法采用称为反射的操作迭代进行的直接搜索方法。遗传算法的交叉与单纯形法的反射操作类似，群体的规模与形成单纯形的端点数类似。模拟退火法具有跳出局部解的能力，它比爬山法有所改进，通过温度降低工序的仿真，动态地对选择概率实施控制。遗传算法的选择操作是在个体适应度基础上以概率方式进行的，在概率选择方式上与模拟退火法有些相似。

自治分布式系统(autonomous distributed system)包括个体的自律性、个体(部分)之间相互作用的非决定性、有序的形态及对环境变化的适应性等。类似的概念还有全息系统(holosystem)、多 agent 系统(multiagent system)等。在遗传算法中，各个个体由其基因型结构决定其表现型性状，主要的操作属于非确定性过程，其中交叉为个体间的相互作用，根据环境中的适应度取值的不同，劣质的个体被淘汰，优秀的个体得以保留，在该选择作用下自适应性也得到体现，同时在种群中也形成一定程度的有序状态。随机搜索特性使得种群保持了一定的分散性，进化选择机制又通过环境下适应度评价的方法，完成种群的自适应优化过程。所以遗传算法也可以从自律分布系统的角度来进行分析和研究。遗传算法具有丰富的动态特性，从数学机理上加以探讨，有助于遗传算法的理论研究和应用。

本章将较全面地介绍遗传算法的基本数学理论和分析工具，包括验证基本遗传算法(SGA)的有效性的模式定理、分析遗传算法过程的 Walsh 模式变换方法、遗传算法的欺骗问题以及遗传算法的动态分析工具 —— Markov 链分析。

3.1 模式定理

3.1.1 模式

遗传算法的执行过程中包含了大量的随机性操作，因此有必要对其数学机理进行分析，为此首先引入“模式”(schema)这一概念。

我们将种群中的个体即基因串中的相似样板称为“模式”，模式表示基因串中某些特征位相同的结构，因此模式也可解释为相同的构形。它描述的是一个串的子集，在二进制编码的串中，模式是基于三个字符集(0, 1, *)的字符串，符号 * 代表任意字符，即 0 或 1。例如模式 *1* 描述了一个四个元的子集{010, 011, 110, 111}。

对于二进制编码串，当串长为 l 时，共有 3^l 个不同的模式，遗传算法中串的运算实际上是

模式的运算。如果各个串的每一位按等概率生成 0 或 1，则规模为 n 的种群模式总数的期望值为

$$\sum_{i=0}^{l} C_l^i 2^i (1-(1-(1/2)^i)^n) \tag{3.1}$$

种群最多可以同时处理 $n \cdot 2^l$ 个模式。从图 3.1 可以看出其内含并行性(implicit parallelism)。如果独立地考虑种群中的各个串，则仅能得到 n 条信息。然而，当把适应值与各个串结合起来考虑，发掘串群体的相似点，我们就得到大量的信息来帮助指导搜索，相似点的大量信息包含在规模不大的种群中。

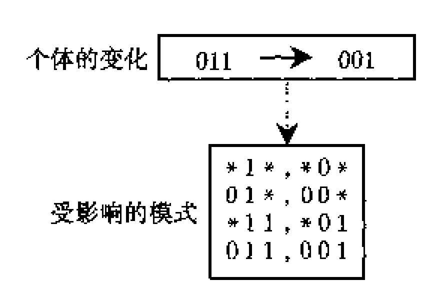


图 3.1 内含并行性

遗传算法是如何利用这些信息的呢？必须考察选择、交叉和变异对模式的作用效果。模式定理给出了这一问题的答案。

3.1.2 模式阶和定义距

所有的模式并不是以同等机会产生的。有些模式比起其他模式更确定，如与模式 0 * * * * * 相比，模式 0 1 1 * * 1 在相似性方面有更明确的表示。有些模式的跨度要比其他的长，如与模式 1 * 1 * * * 相比，1 * * * * 1 要跨越整个串长更大的部分。为了定量地描述模式，我们介绍模式中包含的两个重要参数：模式阶(schema order)和定义距(defining length)。

定义 3.1 模式 H 中确定位置的个数成为模式 H 的模式阶，记作 $O(H)$。例如，$O(0\ 1\ 1\ *\ 1\ *)=4$。

模式阶用来反映不同模式间确定性的差异，模式阶数越高，模式的确定性就越高，所匹配的样本个数就越少。

定义 3.2 模式 H 中第一个确定位置和最后一个确定位置之间的距离称为模式的定义距，记作 $\delta(H)$。例如，$\delta(011*1**)=4$。

在遗传操作中，即使阶数相同的模式，也会有不同的性质，而模式的定义距就反映了这种性质的差异。

3.1.3 模式定理

假定在给定时间步 t，一个特定的模式 H 有 m 个代表串包含在种群 $A(t)$ 中，记为 $m=m(H,t)$，在再生阶段，每个串根据它的适应值进行复制，一个串 Ai 的再生概率为

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{3.2}$$

当采用非重叠的 n 个串的种群代替种群 $A(t)$，可以得到下式：

$$m(H,t+1) = m(H,t) \cdot n \cdot \frac{f(H)}{\sum_{j=1}^{n} f_j} \tag{3.3}$$

其中 $f(H)$ 是在时间步 t 表示模式 H 的串的平均适应度，整个种群的平均适应度可记成：

$$\bar{f} = \frac{\sum_{j=1}^{n} f_j}{n} \tag{3.4}$$

故在基本遗传算法的结构条件下，若遗传操作只选择转移到下一世代的话，则下式成立：

$$m(H,t+1)=m(H,t)\frac{f(H)}{\bar{f}} \tag{3.5}$$

这表明，一个特定的模式按照其平均适应度值与种群的平均适应度值之间的比率生长。换言之，那些适应度值高于种群平均适应值的模式在下一代中将会有更多的代表串，而对于适应度值低于种群平均适应值的模式，它们在下一代中的代表串将会减少。假设从 $t=0$ 开始，某一特定模式适应度值保持在种群平均适应度值以上一个 $c\bar{f}$，c 为一常数，则模式的选择生长方程变为

$$m(H,t+1)=m(H,t)\frac{(\bar{f}+c\bar{f})}{\bar{f}}=(1+c)\cdot m(H,t)=(1+c)^t\cdot m(H,0) \tag{3.6}$$

(3.6)式表明，在种群平均值以上(以下)的模式将按指数增长(衰减)的方式被复制。在一定程度上这种复制算子在种群中并行地采样，许多不同的模式按照相同的规则增长或衰减。仅仅靠复制过程无助于检测搜索空间中新的区域，因为复制并没有搜索新的相似点。因而需要采取交叉操作，交叉是两个串之间随机地进行信息交换。这里仅考虑单点交叉的场合。

为了搞清楚模式受交叉影响的方式和程度，我们以一个串长为 7 的特定串和包含在其中的两个具有代表性的模式为例：

$$\begin{aligned} A\ &= 0\ 1\ 1\ 1\ 0\ 0\ 0 \\ H_1 &= *\ 1\ *\ *\ *\ *\ 0 \\ H_2 &= *\ *\ *\ 1\ 0\ *\ * \end{aligned}$$

设串 A 被选择来进行交叉，假设随机地选择一个交叉位置(在串长为 7 时仅有 6 个可选位置)，如果选定位置在 3 和 4 之间，考察一下交叉对模式 H_1 和 H_2 的作用。其中交叉位置用“|”标记。

$$\begin{aligned} A\ &= 0\ 1\ 1\ |\ 1\ 0\ 0\ 0 \\ H_1 &= *\ 1\ *\ |\ *\ *\ *\ 0 \\ H_2 &= *\ *\ *\ |\ 1\ 0\ *\ * \end{aligned}$$

显然，如果 A 的交叉对象在模式 H_1 的确定位置上与 A 不同，模式 H_1 将被破坏。而对于相同的交叉位置，模式 H_2 将保留到一个子代串中。虽然这里取的交叉位置比较特别，但是很明显，在交叉过程中模式 H_1 比模式 H_2 更不易生存，这是因为交叉点一般更易落在距离最远的确定位置之间。一般地，模式 H 被破坏的概率为 $\delta(H)/(l-1)$，模式 H 生存概率为 $1-\delta(H)/(l-1)$。模式 H_1 被破坏的概率为 5/6(生存概率为 1/6)；模式 H_2 被破坏的概率为 1/6(生存概率为 5/6)。考虑到交叉本身是以随机方式进行的，即以概率 p_c 进行的交叉，因此对于模式 H 的生存概率可以计算如下：

$$p_s\geqslant 1-p_c\cdot\frac{\delta(H)}{l-1} \tag{3.7}$$

同时考虑选择、交叉操作对模式的影响，由于选择与交叉操作是不相关的，可以得到子代模式的估计：

$$m(H,t+1)\geqslant m(H,t)\cdot\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}\right] \tag{3.8}$$

(3.8)式表明,模式增长或衰减依赖于两个因素,一个因素是模式的适应值是在平均适应值之上还是在平均适应值之下,另一个因素是模式具有相对长还是相对短的定义距。那些既在种群平均适应度值之上同时又具有短的定义距的模式将按指数增长率被采样。

下面再考察变异操作对模式的影响。变异操作是以概率 p_m 随机地改变一个位上的值,为了使得模式 H 可以生存下来,所有特定的位必须存活。因为单个等位基因存活的概率为$(1-p_m)$,并且由于每次变异都是统计独立的,因此,当模式 H 中$O(H)$个确定位都存活时,这时模式 H 才被保留下来,存活概率为

$$(1-p_m)^{O(H)} \approx 1-O(H)\cdot p_m \qquad (p_m \ll 1) \tag{3.9}$$

因此,在考虑选择、交叉和变异操作的作用下,一个特定模式在下一代中期望出现的数目可以近似地表示为

$$m(H,t+1) \geqslant m(H,t)\cdot \frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}-O(H)p_m\right] \tag{3.10}$$

式中,$m(H,t+1)$ —— 表示在 $t+1$ 代种群中存在模式 H 的个体数目;

$m(H,t)$ —— 表示在 t 代种群中存在模式H 的个体数目;

$f(H)$ —— 表示在 t 代种群中包含模式H 的个体平均适应度;

$\bar{f}$ —— 表示在 t 代种群中所有个体的平均适应度;

l —— 表示个体的长度;

p_c—— 表示交叉概率;

p_m—— 表示变异概率。

对于 k 点交叉的场合,上式可以作如下改变(证明从略):

$$m(H,t+1) \geqslant m(H,t)\cdot \frac{f(H)}{\bar{f}}\left[1-p_c\frac{C_l^k-C_{l-1-\delta(H)}^k}{C_{l-1}^k}-O(H)p_m\right] \tag{3.11}$$

综上所述,我们可以得到遗传算法的一个非常重要的结论——模式定理(schema theorem)。

定理 3.1 在遗传算于选择、交叉、变异的作用下,具有低阶、短定义距以及平均适应度高于种群平均适应度的模式在子代中呈指数增长。

模式定理是遗传算法的基本理论,保证了较优的模式(遗传算法的较优解)的数目呈指数增长,为解释遗传算法机理提供了一种数学工具。

3.1.4 积木块假设

在模式定理中所指的具有低阶、短定义距以及平均适应度高于种群平均适应度的模式被定义为积木块(building block)。它们在遗传算法中很重要,在子代中呈指数增长,在遗传操作下相互结合,产生适应度更高的个体,从而找到更优的可行解。

积木块假设(building block hypothesis) 遗传算法通过短定义距、低阶以及高平均适应度的模式(积木块),在遗传操作作用下相互结合,最终接近全局最优解。

满足这个假设的条件比较简单,包括两方面:

① 表现型相近的个体,其基因型类似;

② 遗传因子间相关性低。

下面考虑一个整数解的最优化问题,采用 4 位二进制基因码。表现型 7 对应基因型

0111;8 对应 1000。假定 7 是问题的最优解,种群中有 8 对应的个体 1000,用一般的遗传算子不易生成最优解 7 对应的个体 0111,但采用逆位(inversion)操作,则 1000 很容易变为 0111。图 3.2 表示了种群中的积木块。

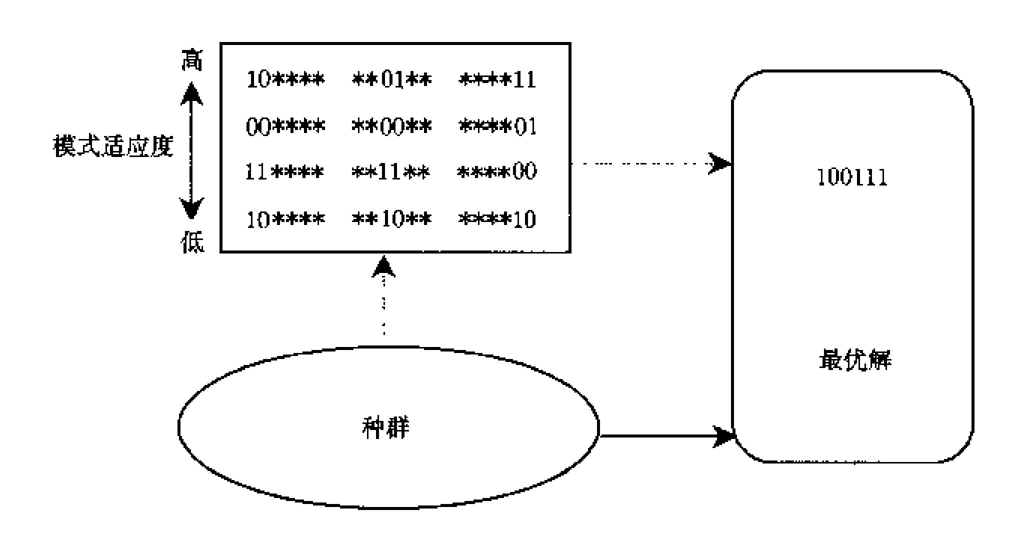


图 3.2　适于逆位操作的积木块

目前大量的实践支持积木块假设,它在许多领域内都取得成功,例如平滑多峰问题、带干扰多峰问题以及组合优化问题等。模式定理保证了较优模式(遗传算法的较优解)的样本数呈指数增长,从而满足了求最优解的必要条件,即遗传算法存在找到全局最优解的可能性;而积木块假设指出,遗传算法具备寻找全局最优解的能力,即积木块在遗传算子的作用下,能生成低阶、短距、高平均适应度的模式,最终生成全局最优解。

虽然模式定理在一定意义上解决了基本遗传算法(SGA)的有效性,但它仍有以下的缺点:

① 模式定理只对二进制编码适用,对其他编码方案尚没有相应的结论成立。

② 模式定理只给出了在下世代包含模式 H 的个体数的下限。我们无法据此推断算法的收敛性。

③ 模式定理没有解决算法设计中控制参数选取等问题。

3.2　Walsh 模式变换

在数字信号处理中,Walsh 函数(Walsh, 1923 年)是基函数完备的正交集,可提供类似傅里叶变换的表示方式。A. D. Bethke 的博士论文"作为函数优化器的遗传算法"(密歇根大学,1980 年),首次提出了应用 Walsh 函数进行遗传算法的模式处理,并引入 Walsh 模式变换的概念,采用 Walsh 函数的离散形式有效地计算模式的平均适应度,从而对遗传算法的优化过程的特征进行分析。在 Bethke 的工作的基础上,Goldberg(1989 年)进一步发展了这一分析方法。

3.2.1　Walsh 函数

Walsh 函数的基函数,只有两个值 +1 和 −1。Bethke 试图用这些基函数来构造遗传算法困难程度的函数,为此将 Walsh 函数离散化,构成值域$\{0,1\}^l$ 的正交基。对 $j=\{0,1,\cdots,2^l-1\}$的 l 位二进制编码统一定义为$j=\langle j_{l-1}j_{l-2}\cdots j_0\rangle$。

定义 3.3(Walsh 函数) Walsh 函数 $\Psi_j(X)(X=(x_{l-1}x_{l-2}\cdots x_0),\ x_i=\{0,1\},\ i=0,1,\cdots,l-1)$定义为

$$\Psi_j(X)=\prod_{i=0}^{l-1}(1-2x_i)^{j_i} \tag{3.11}$$

即 $\Psi_j(X)=(-1)^{\eta},\ \eta=\sum_{i=0}^{l-1}j_ix_i$。例如,$x$ 为 3 位时 Walsh 函数 $\Psi_0(X),\Psi_1(X),\cdots,\Psi_7(X)$ 的计算式列于表 3.1。

表 3.1 Walsh 函数

$\Psi_0(X)$	$\Psi_1(X)$	$\Psi_2(X)$	$\Psi_3(X)$
1	$1-2x_1$	$1-2x_2$	$(1-2x_1)(1-2x_2)$
$\Psi_4(X)$	$\Psi_5(X)$	$\Psi_6(X)$	$\Psi_7(X)$
$1-2x_3$	$(1-2x_1)(1-2x_3)$	$(1-2x_2)(1-2x_3)$	$(1-2x_1)(1-2x_2)(1-2x_3)$

Walsh 函数的离散值计算,如图 3.3 所示,图中阴影部分位置的函数值为 1,空白部分为 0。

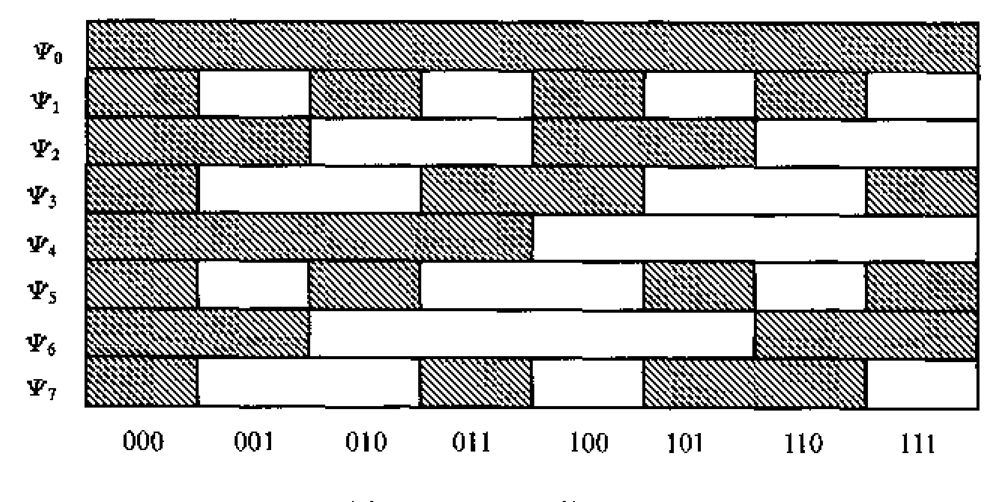


图 3.3 Walsh 函数的图示

定理 3.2(Walsh 函数的正交性) Walsh 函数的基函数为正交函数,满足

$$\sum_{X=0}^{2^l-1}\Psi_i(X)\Psi_j(X)=\begin{cases}2^l, & i=j\\0, & i\neq j\end{cases} \tag{3.12}$$

定理 3.3(任意函数的 Walsh 函数展开式) 设 $X=(x_{l-1}x_{l-2}\cdots x_0),\ x_i=\{0,1\},\ i=0,1,\cdots,l-1$,则 X 定义域为$\{0,1,\cdots,2^l-1\}$的任意函数 $f(X)$可以展开表示成 Walsh 多项式:

$$f(X)=\sum_{X=0}^{2^l-1}\omega_j\Psi_j(X) \tag{3.13}$$

式中 X 为一长度为 l 的二进制位串,ω_j 为 Walsh 系数

$$\omega_j=\frac{1}{2^l}\sum_{X=0}^{2^l-1}f(X)\Psi_j(X) \tag{3.14}$$

3.2.2 用 Walsh 函数表示模式平均适应度

考虑模式 H 的平均适应度

$$f(H) = \frac{1}{|H|}\sum_{X\in H} f(X) = \frac{1}{|H|}\sum_{X\in H}\sum_{j=0}^{2^l-1}\omega_j\Psi_j(X)$$

$$= \sum_{j=0}^{2^l-1}\omega_j\frac{1}{|H|}\sum_{X\in H}\Psi_j(X) \tag{3.15}$$

设 $S(H,j)=\frac{1}{|H|}\sum\limits_{X\in H}\Psi_j(X)$，其值域为 $\{-1,0,1\}$。例如，当 $l=3$ 时计算如表 3.2 所示，表中 $y_i=1-2x_i$。

表 3.2 $S(H,j)$的值

H \ J	0 1	1 y_0	2 y_1	3 y_0y_1	4 y_2	5 y_0y_2	6 y_1y_2	7 $y_0y_1y_2$
* * *	1	0	0	0	0	0	0	0
* * 0	1	1	0	0	0	0	0	0
* * 1	1	−1	0	0	0	0	0	0
* 0 *	1	0	1	0	0	0	0	0
* 0 0	1	1	1	0	0	0	0	0
* 0 1	1	−1	1	−1	0	0	0	0
* 1 *	1	0	−1	0	0	0	0	0
* 1 0	1	1	−1	−1	0	0	0	0
* 1 1	1	−1	−1	1	0	0	0	0
0 * *	1	0	0	0	1	0	0	0
0 * 0	1	1	0	0	1	1	0	0
0 * 1	1	−1	0	0	1	−1	0	0
0 0 *	1	0	1	0	1	0	1	0
0 0 0	1	1	1	1	1	1	1	1
0 0 1	1	−1	1	−1	1	−1	1	−1
0 1 *	1	0	−1	0	1	0	−1	0
0 1 0	1	1	−1	−1	1	1	−1	−1
0 1 1	1	−1	−1	1	1	−1	−1	1
1 * *	1	0	0	0	−1	0	0	0
1 * 0	1	1	0	0	−1	−1	0	0
1 * 1	1	−1	0	0	−1	1	0	0
1 0 *	1	0	1	0	−1	0	−1	0
1 0 0	1	1	1	1	−1	−1	−1	−1
1 0 1	1	−1	1	−1	−1	1	−1	1
1 1 *	1	0	−1	0	−1	0	1	0
1 1 0	1	1	−1	−1	−1	−1	1	1
1 1 1	1	−1	−1	1	−1	1	1	−1

由于 $f(H)=\sum\limits_{j\in J(H)}\omega_j S(H,j)$，$J(H)$为 H 包含的模式序号，例如，$H=01*$ 所包含的模式有 $h_0=***$，$h_2=*f*$，$h_4=f**$，$h_6=ff*$，则 $J(H)=\{0,2,4,6\}$。模式 H 的平均适应

度$f(H)$是 Walsh $\{\omega_i\}$与$\{-1,0,1\}$求积的和。

综上所述,模式平均适应度的值计算分为三个步骤:

① 计算 $f(X)$的值;

② 由 Walsh 变换(3.15)式求 Walsh 系数;

③ 根据 Walsh 系数与模式变换表,计算模式平均适应度的值。

Goldberg 对第③步作了改进,采用类似快速傅里叶变换 FFT 的方法,由函数值求取 Walsh 系数,如图 3.4 所示。

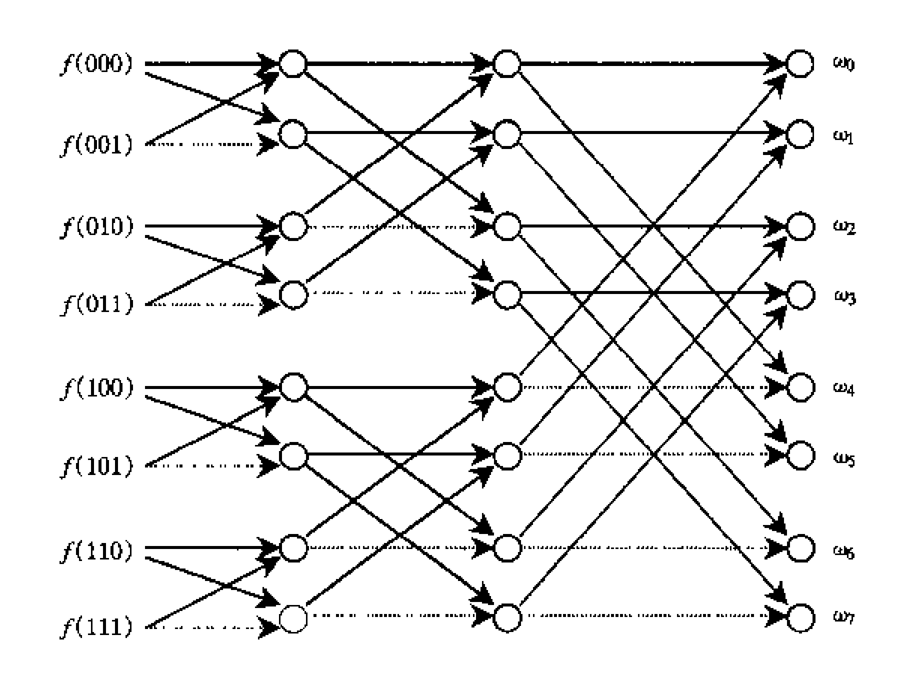


图 3.4 函数值与 Walsh 系数的关系

图 3.5 表示了模式 1 * * , * * 0, 1 * 0 对应的位串的分布。

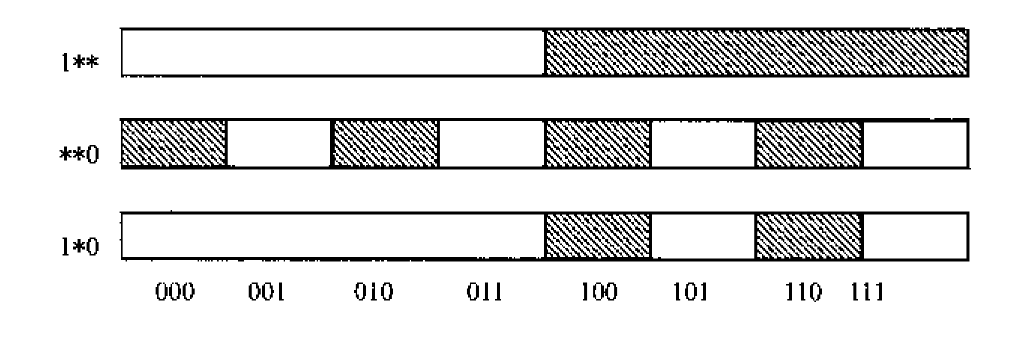


图 3.5 模式对应的位串

3.2.3 Walsh 系数与异位显性(epistasis)

模式定理认为,具有低阶、短定义距以及平均适应度高于种群平均适应度的模式,即所谓的积木块,对遗传算法非常重要,如果很容易地获得,说明遗传算法的效果好。

我们知道,0 阶模式代表了种群中所有的个体,模式 * * 1, * 0 * 等表示 1 阶模式的积木块。根据 Walsh 模式变换可以由低阶的模式适应度估计高阶模式的适应度值。

上述的例子中，0 阶估计 $f(* * *)$是由式(3.13)和(3.14)很容易求得的。这里 $f(* * *)=\omega_0$，由 0 阶估计推算 1 阶估计的方法是：如果确定位在上位，取适应度的偏差值为 $\pm\omega_4$；如果确定位在中位，取适应度的偏差值为 $\pm\omega_2$；如果确定位在下位，取适应度的偏差值为 $\pm\omega_1$。

$$f'(1 * *)=\omega_0-\omega_4$$

$$f'(0 * *)=\omega_0+\omega_4$$

2 阶和 3 阶的适应度估计值也可以类似地求得，例如：

$$f'(1\ 1\ *)=\omega_0-\omega_2-\omega_4$$

$$f'(0\ 1\ *)=\omega_0-\omega_2+\omega_4$$

$$f'(1\ *\ 1)=\omega_0-\omega_1-\omega_4$$

$$f'(*\ 1\ 1)=\omega_0-\omega_1-\omega_2$$

$$f'(0\ 1\ 1)=\omega_0-\omega_1-\omega_2+\omega_4$$

上述估计值与准确值存在一些差别，如根据表 3.2 得到：

$$f(1\ 1\ *)=\omega_0-\omega_2-\omega_4+\omega_6$$

$$f(0\ 1\ *)=\omega_0-\omega_2+\omega_4-\omega_6$$

$$f(1\ *\ 1)=\omega_0-\omega_1-\omega_4+\omega_5$$

偏差值的 walsh 系数项的序号是：将模式中确定位均置为 1，不确定位 * 均置 0，得到一个二进制数的数值；偏差值的符号根据模式中确定位 1 的总个数的奇偶性确定。

我们将模式适应度估计值与准确值的差异称为异位显性(epistasis)，在生物学中异位显性是指一个基因对另一个非等位基因表现的显性现象。如果异位显性很大，便不能解释仅借助低次模式的积木块生成最优解的机制。

假设 $f(x_3x_2x_1)=x_3-2x_2-x_1, x_i\in\{0,1\}$，模式的 walsh 系数计算如表 3.3 所示。

如果异位显性为正值，表示由低阶积木块生成较高适应度的高阶模式是困难的；相反，如果异位显性为负值，表示由低阶积木块生成较高适应度的高阶模式是容易的。也就是说，对于最优解对应的可能模式而言，低阶估计值比其实际计算值越高时说明其获得最优解可能性越大，反之获得最优解的可能性越小。这与模式定理的结论是一致的。

表 3.3　Walsh 系数

j	$x_3x_2x_1$	$f(x_3x_2x_1)$	ω_j
0	000	0	-0.5
1	001	-1	0.5
2	010	-2	1
3	011	-3	0
4	100	1	-0.5
5	101	0	0
6	110	-1	0
7	111	-2	0

为了说明 Walsh 模式变换的应用，作为示例，我们来分析另一种与普通二进制编码不同的 Gray 编码方式的适用性问题。

Grey 编码$(g_{l-1}g_{l-2}\cdots g_0)$与普通二进制编码$(b_{l-1}b_{l-2}\cdots b_0)$的关系，可以用下式表示：

$$g_k=\begin{cases}b_{l-1}, & k=l-1\\ b_{k+1}\oplus b_k, & k\leqslant l-2\end{cases}\tag{3.16}$$

上式中⊕为排斥律逻辑谓词。Grey 编码与普通二进制编码的逆关系可以表示为：

$$b_k = \sum_{i=k}^{l-1} g_i \pmod 2 \qquad k = 0,1,\cdots,l-1 \tag{3.17}$$

表3.4为一个三位二进制编码与其对应的Grey编码的示例。

表3.4 二进制编码与其对应的Grey编码

十进制数	二进制编码	Grey编码
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

对于某些优化问题,采用Grey编码比二进制编码优越。下面从模式分析的角度作一剖析。例如,对于最小化问题 $f(x)=(x-4)^2$,分别采用二进制编码和Grey编码,分别计算适应度值和Walsh系数,如表3.5所示。表3.6列出了模式的评价值。

表3.5 Walsh系数

Walsh函数		二进制编码			Grey编码		
编码	编号	x	$f(x)$	ω	x	$f(x)$	ω
000	0	0	16	44	0	16	44
001	1	1	9	4	1	9	8
010	2	2	4	8	3	1	32
011	3	3	1	8	2	4	16
100	4	4	0	16	7	9	16
101	5	5	1	16	6	4	0
110	6	6	4	32	4	0	8
111	7	7	9	0	5	1	4

表3.6 模式评价值

二进制编码		Grey编码	
H	$f(H)$	H	$f(H)$
* * 0	48	* * 0	48
* 0 *	52	* 1 *	12
* 0 0	64	* 1 0	4
1 * *	28	1 * *	28
1 * 0	16	1 * 0	36
1 0 *	4	1 1 *	4
1 0 0	0	1 1 0	0

从模式评价值可知，Grey编码的中间位突出了最优模式。比较定义距为1的模式 $ff*$ 和 $*ff$ 的评价值，Grey编码较普通二进制编码优越；比较2阶以上的 $\omega_j(j\neq 1,2,4)$ 的值，一般Grey编码比较小。这些说明了Grey编码是支持积木块假设的。

3.3　非均匀Walsh模式变换

上一节中，我们介绍的Walsh模式中未考虑个体数的分布问题，模式的平均评价值计算时，假定个体服从均匀分布，这不适用于遗传算法初期个体随机产生的情况。因此有必要考虑模式个体数的非均匀Walsh模式变换。

设个体的二进制编码 $X=(x_{l-1}x_{l-2}\cdots x_0)$，$x_i\in\{0,1\}$。模式 $H=(h_{l-1}h_{l-2},\cdots,h_0)$，$h_i\in\{0,1,*\}$。$f(X)$为 X 的适应度函数，$P(X)$为 X 在种群 Ω 中所占的比例。

$$\sum_{X\in\Omega}P(X)=1 \tag{3.18}$$

属于模式 H 的个体数比例为 $P(H)$

$$P(H)=\sum_{X\in H}P(X) \tag{3.19}$$

模式 H 的适应度平均值为

$$f(H)=\frac{\sum_{X\in H}f(X)P(X)}{P(H)} \tag{3.20}$$

由于属于模式 H 的个体数目为 $2^{O(H)}$，假设 $P(H)=2^{-O(H)}$，这时 $f(H)$的评价值为

$$\bar{f}(H)=f(H)\frac{P(H)}{2^{-O(H)}} \tag{3.21}$$

将式(3.20)代入(3.21)，可得

$$\bar{f}(H)=\frac{1}{2^{l-O(H)}}\sum_{X\in H}f(X)P(X)2^l \tag{3.22}$$

因此，将下式

$$\bar{f}(X)=f(X)P(X)2^l \tag{3.23}$$

代替Walsh模式变换公式中的 $f(X)$，即可得到非均匀Walsh变换。因而，考虑个体数分布，进行模式分析也是可行的。

3.4　欺骗问题

在遗传算法中，将所有妨碍评价值高的个体生成从而影响遗传算法正常工作的问题统称为欺骗问题(deceptive problem)。遗传算法运行过程具有将高于平均适应度、低阶和短定义距的模式重组成高阶模式的趋势。如果在低阶模式中包含了最优解的话，则遗传算法就可能找出它来。但是低阶、高适应度的模式可能没有包含最优串的具体取值，于是遗传算法就会收敛到一个次优的结果。下面给出有关欺骗性的概念。

定义3.4(竞争模式)　若模式 H 与 H' 中，$*$ 的位置完全一致，但任一确定位的编码均不

同，则称 H 与 H' 互为竞争模式。

定义3.5(欺骗性) 假设 $f(X)$的最大值对应的 X 集合为X^*，H 为一包含X^*的 m 阶模式。H 的竞争模式为H'，而且 $f(H)>f(H')$，则 f 为 m 阶欺骗。

在上述定义中，当 $m=l$ 时，由于模式中不包含 *，自然不存在欺骗性问题。

例如，对于一个 3 位二进制编码的模式，如果 $f(111)$为最大值时，下列 12 个不等式中任意一个不等式成立，则存在欺骗性问题。

模式阶数为 1 时 $f(**1)<f(**0)$，$f(*1*)<f(*0*)$，$f(1**)<f(0**)$

模式阶数为 2 时 $f(*11)<f(*00)$，$f(1*1)<f(0*0)$，$f(11*)<f(00*)$

$f(*11)<f(*01)$，$f(1*1)<f(0*1)$，$f(11*)<f(01*)$

$f(*11)<f(*10)$，$f(1*1)<f(1*0)$，$f(11*)<f(10*)$

阶数 1 的条件分别与 $\omega_1>0, \omega_2>0, \omega_3>0, \omega_4>0$ 等价。

对于阶数 2 的条件，例如 $f(1*1)<f(0*1)$可化为

$$\omega_0-\omega_1-\omega_4+\omega_5<\omega_0-\omega_1+\omega_4-\omega_5$$

即 $\omega_4<\omega_5$。其他条件也可作类似的推导和化简。

3.4.1 欺骗函数的类型

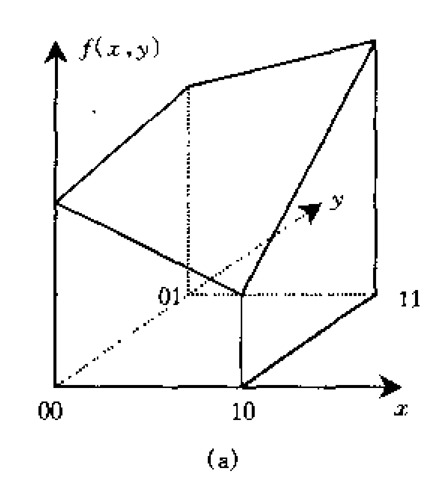
Goldberg 曾研究用适应度函数的非单调性来研究欺骗性问题。考虑一个 2 位二进制最大化问题，假定“11”对应最优解，若 $H(0*)>H(1*)$，则欺骗性存在。该条件可化为

$$\frac{f(00)+f(01)}{2}>\frac{f(10)+f(11)}{2} \tag{3.24}$$

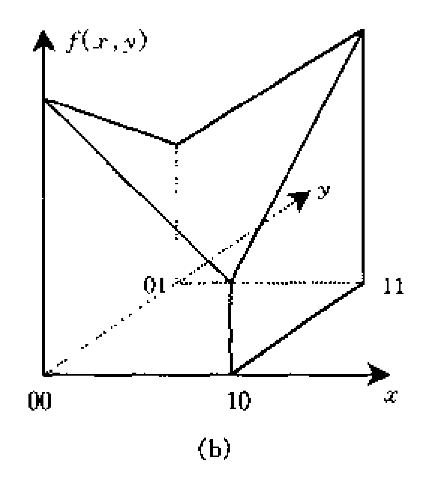
设 $r=f(11)/f(00)$，$c=f(01)/f(00)$，$c'=f(10)/f(00)$，则(3.24)式约简为

$$r<1+c-c' \tag{3.25}$$

Goldberg 将 $c>1$ 的情况称为Ⅰ类欺骗问题，$c\leqslant1$ 的情况称为Ⅱ类欺骗问题，如图 3.6 所示。对遗传算法而言，Ⅱ类欺骗问题的求解比Ⅰ类欺骗问题更困难。在图 3.7 中，若以(f_{00}，f_{00})为原点时，在第一象限，因为适应度随每个遗传因子单调增加，叫单调问题。在二、四象限成为Ⅰ类欺骗问题的存在领域，在第三象限成为Ⅱ类欺骗问题的存在领域。Ⅰ类和Ⅱ类虚拟



(a)



(b)

图 3.6 欺骗问题的两种类型

(a) Ⅰ类欺骗问题； (b) Ⅱ类欺骗问题

问题应该称为非单调问题，在非单调问题中同时存在欺骗和非欺骗问题。

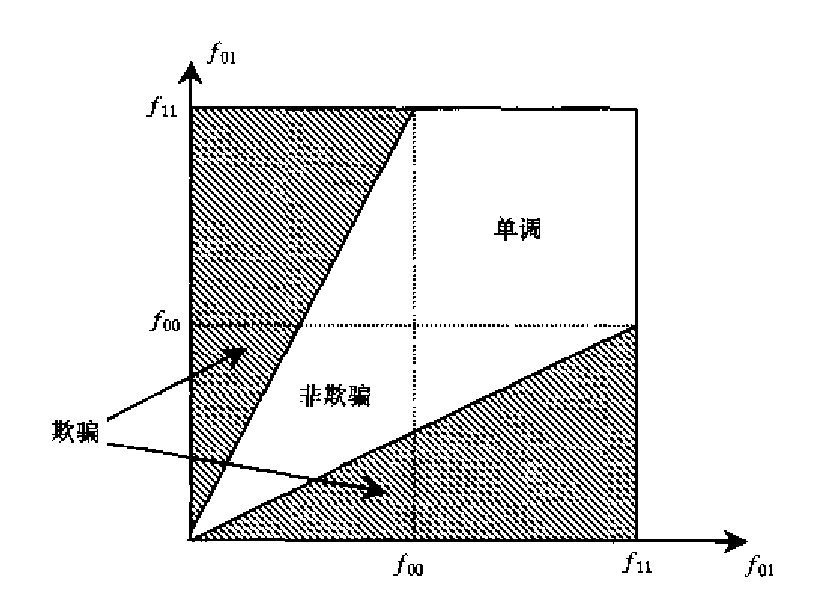


图 3.7 2 位两个个体的欺骗示域图

过去，将适应度函数的非单调问题与欺骗问题同等看待，认为遗传算法只有在单调问题里有效。但是，如果单调问题不使用遗传算法或者不使用概率搜索，一般的搜索法可能是适用的，没有遗传算法存在的必要。即使是非单调，只有存在需要高机能交叉操作(非单调且非欺骗问题)，才能使遗传算法存在有意义，这不外乎使交叉操作成为遗传算法本质作用的一个证明。

3.4.2 欺骗性的化解

遗传算法中欺骗性的产生往往与适应度函数确定和调整、基因编码方式相关。下面以合适的编码方式为例，说明如何化解和避免欺骗问题。

一个 2 位编码的适应度函数

$$f(x) = 4 + \frac{11}{6}x - 4x^2 + \frac{7}{6}x^3 \tag{3.26}$$

采用二进制编码，计算个体的函数值如表 3.7 所示。这时，存在所谓的 Ⅱ 类欺骗问题。

采用 Grey 编码，计算个体的函数值如表 3.8 所示。这时，Ⅱ 类欺骗问题化解为 Ⅰ 类欺骗问题。最优解编码为 10。

表 3.7 二进制编码函数值

编码	对应整数解	函数值
00	0	4
01	1	3
10	2	1
11	3	5

表 3.8 Grey 编码函数值

编码	对应整数解	函数值
00	0	4
01	1	3
11	2	1
10	3	5

下面的例子说明了采用适当的适应度函数调整方法可以避免欺骗问题。

设目标函数 $g(00)=128$, $g(01)=1$, $g(10)=g(11)=32$，如果适应度函数 $f(x)=g(x)$，

则 $f(0*)=64.5, f(1*)=32$。这时，存在欺骗问题。

如果用适应度函数的调整方法，$f(x)=\log_2 g(x)$，则

$$f(00) - 7, f(01) - 0, f(11) - f(10) - 5$$

得到 $f(0*)=3.5, f(1*)=5$，从而不会产生欺骗问题。

3.4.3　遗传算法的困难问题

我们将采用基本的遗传操作包括选择和再生、交叉与变异，以及标准的操作参数，进行模拟进化的过程求解最优解很容易的场合，称为遗传算法的容易问题；反之，称为遗传算法的困难问题。遗传算法的欺骗性并不一定是导致遗传算法的困难问题，同样的非欺骗性并不能说明会产生遗传算法的容易问题。遗传算法的欺骗性与遗传算法的困难性不存在等价的关系，这是由于遗传算法的欺骗性是从静态的超平面分析中给出的，并且假定个体数无偏差，而遗传算法的困难性来源于不适当的问题表示、交叉和变异的扰动作用、有限的种群大小、复杂的多模型状态图等。

例如，下面的欺骗问题，却是遗传算法的容易问题，如图 3.8 所示。

$$\max_{0\leqslant x_1, x_2\leqslant 1} f(x_1, x_2) = \begin{cases} x_1^2 + 10x_2^2, & x_2 < 0.995 \\ 2(1 - x_1)^2 + 10x_2^2, & \text{其他} \end{cases} \tag{3.27}$$

这里 x_1, x_2 分别用 10 位二进制编码表示。很明显，最优解为(0, 1)。

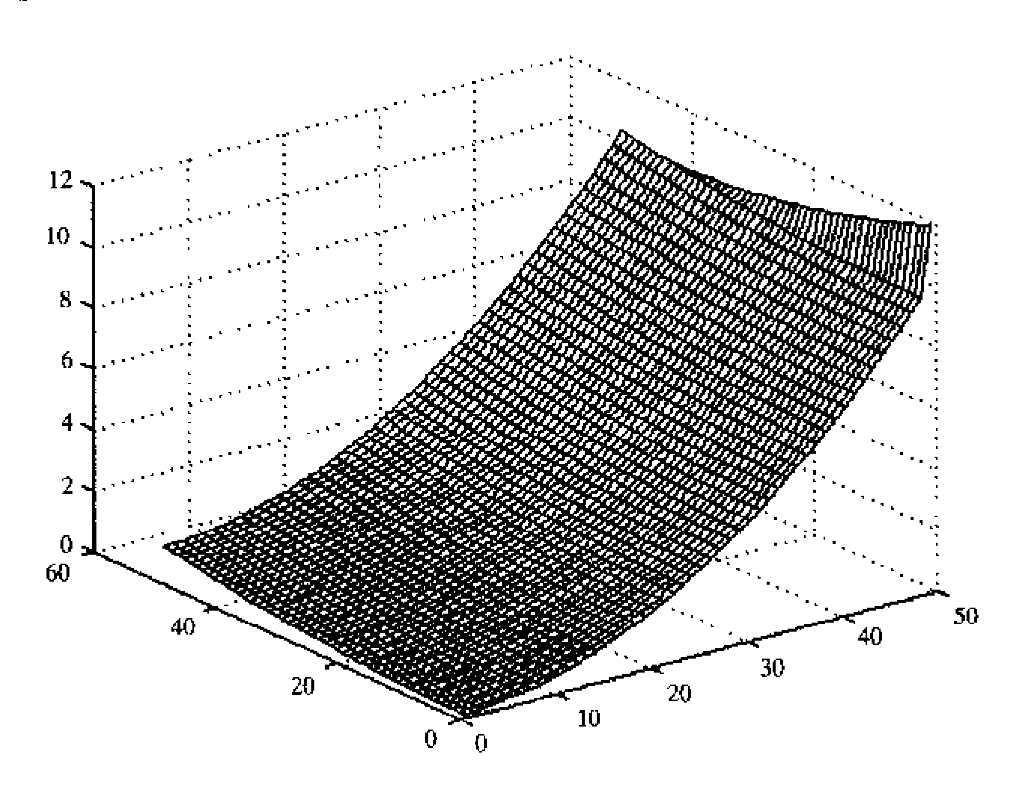


图 3.8　一个遗传算法容易产生欺骗问题的函数图形

模式(1111111111 # # # # # # # # # #)包含最优解，它比问题解(1, #)的适应度高，因而存在欺骗问题。但对遗传算法而言却是容易问题。

下面的问题属于非欺骗问题，但却是遗传算法的困难问题。

$$\max_{0\leqslant x\leqslant 1} f(x) = \begin{cases} 2^{L+1}, & x = 0 \\ x^2, & \text{其他} \end{cases} \tag{3.28}$$

上式中 x 的二进制码长为 L。据 T. Kuo(1993年)对这类遗传算法困难问题的研究,可以采用破坏性选择方法(disruptive selection)的遗传算法来解决。

3.5 遗传算法动态分析

从随机过程和数理统计角度探讨遗传算法较为一般的规律,有助于较好地把握遗传算法的特性,以提高求解效率和改善求解效果。铃木(1995年)从 Markov 链的角度分析了基本遗传算法(SGA)的统计规律,并得出了一些有意义的结论。

SGA 的当前种群只与前一代种群有关,因此 SGA 可以用一个 Markov 链来描述。

定义 3.6 (有限 Markov 链) 设 $\{x_t, t\geqslant 0\}$ 是一列取值为有限状态空间 $S=\{s_1, s_2, \cdots, s_n\}$ 上的随机变量,若 x_{k+1} 所在的状态只与 x_k 有关,而与 $x_0, x_1, \cdots, x_{k-1}$ 无关,即对于任意 $k\geqslant 0$ 的正整数 $i_0, i_1, \cdots, i_k, i_{k+1}$,有

$$P(x_{k+1} = s_{i_{k+1}} \mid x_0 = s_{i_0}, \cdots, x_k = s_{i_k}) = P(x_{k+1} = s_{i_{k+1}} \mid x_k = s_{i_k}) \tag{3.29}$$

成立,则称 $\{x_t, t\geqslant 0\}$ 为 Markov 链, $P(x_{t+1}=s_j \mid x_t = s_i)$ 称为在时刻 t 由状态 s_i 转移到状态 s_j 的转移概率,记为 $p_{ij}(t)$。若转移概率与时间无关,则称 Markov 链为齐次的。

SGA 的当前种群只与前一代种群有关,因此 SGA 可以用一个 Markov 链来描述。

令个体 i 的适应度值为 $f(i)$,种群规模记为 M,个体串长为 L,并将个体表示为如下形式:

$$a_{i1}a_{i2}\cdots a_{iL} \in A^L$$

这里, $A=\{0,1,\cdots,\alpha\}(\alpha\geqslant 2)$, a_{ij} 取 c 进制值 $j=1,2,\cdots,L$。每个个体 i 也可记为无符号整数形式 $\sum_{j=1}^{L} a_{ij}\alpha^{L-j}$。每个种群 k 记为向量形式:

$$[Z(0,k), Z(1,k), \cdots, Z(\alpha^L - 1, k)]$$

这里, $Z(i,k)$ 表示标记为 $i=0,1,\cdots,\alpha^L-1$ 的个体出现次数。

为简化讨论起见,令 $\alpha=2$,选择按下列概率 $p(i,k)$ 进行:

$$p(i,k) = \frac{f(i)Z(i,k)}{\sum_{h=0}^{2^L-1} f(h)Z(h,k)} \tag{3.30}$$

交叉概率记为 x,形式为单点交叉,即从 $(1, L-1)$ 中以均匀概率选择交叉点。突变概率为 μ,对个体的位值进行翻转操作。每次保留1个具有最高适应度的个体作为良种,记为 $i^*(k)$。

种群的初始分布可记为

$$q^{(0)} = (q_1^{(0)}\ q_2^{(0)}\ \cdots\ q_N^{(0)})$$

令 Q 表示转移矩阵,则 n 代时的种群为

$$q^{(n)} = (q_1^{(n)}\ q_2^{(n)}\ \cdots\ q_N^{(n)}) = q^{(0)}Q^n \tag{3.31}$$

引理 3.1 存在满足下式的系数 $\rho_{k,v}^{(t)}(t=1,2,\cdots,N)$

$$q_{k,v}^{(n)} = \sum_{t=1}^{N} \rho_{k,v}^{(t)} \lambda_t^n \tag{3.32}$$

当且仅当转移矩阵 $Q=(q_{k,v})$是本原的(不可约和非周期的)或者不可分解的(可约为只有一个非周期递归类的情况),其中

$$1 = |\lambda_1| \geqslant |\lambda_2| \geqslant \cdots \geqslant |\lambda_N|$$

而且 $\lambda_t(t=1,2,\cdots,N)$是转移矩阵的特征值,$\rho_{k,v}^{(t)}$是种群$k$ 的不可约值,并与种群 $v=1,2,\cdots,N$ 的平稳概率$q_v^{(\infty)}$ 相一致。

引理 3.2 在修改的良种策略中,从种群 k 到 v 的转移矩阵 $Q=(q_{k,v})$具有 2^L 个子矩阵 $Q(i)$。$Q(i)$的大小为 $N(i)\times N(i)$, $i=0,1,\cdots,2^L-1$,它的对角元素、对角线右上角的元素均为零,其中,$N(i)$是种群 k 的数量,这里 $i=i^*(k)$。

定理 3.4

$$N(i) = C_{M-1+2^L-i}^{M-1} \tag{3.33}$$

引理 3.3 每一子矩阵 $Q(i)$ ($i=0,1,\cdots,2^L-1$)的 $N(i)$个特征值与矩阵 Q 的 $N=\sum_{i=0}^{2^L-1} N(i)$个特征值是同一的。

定理 3.5 存在满足下式的常数 C,

$$\sum_{k\in K} a_k^{(n)} \geqslant 1 - C|\lambda_*|^n \tag{3.34}$$

其中$|\lambda_*| = \max\limits_{1\leqslant i\leqslant 2^L} \max\limits_{1\leqslant j\leqslant N(S)} |\lambda_{i,j}| < 1$,且 $\lambda_{i,j}(j=1,2,\cdots,N(i))$表示子矩阵 $Q(i)$的 $N(i)$个特征值($i=0,1,\cdots,2^L-1$)。

定理 3.6 对于 λ_*,存在不依赖于变异概率 μ 且满足下式的常数A,

$$|\lambda_*| \leqslant 1 - A\mu^{\delta}(1-\mu)^{L-\delta} \tag{3.35}$$

其中突变阶 δ 是个体i 和$j(<i)$之间的海明(Hamming)距离 $d(i,j)$的极值。海明距离是指两个个体 L 位中位值不一致的个数。突变阶 δ 可被认为是陷入局部搜索的程度。

推论 3.1 如果突变概率 μ 满足$\mu=\delta/L$ 时,$1-A\mu^{\delta}(1-\mu)^{L-\delta}$最小。

从此推论中可以获知,如果变异概率与突变阶之间存在正比关系,$|\lambda_*|$获得最小。

以上基于有限种群模型的 3 个定理,有助于对遗传算法内在机理的探讨,所得的结论适用于良种选择策略(精英主义策略)引导下的遗传算法。在选择操作中保留当前最好解的基本遗传算法(SGA)能以概率收敛到最优解。

参考文献

[1] Forrest S, Mitchell M. What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation. In: Proceedings of the Fourth International Conference on Genetic Algorithms, CA, Morgan Kaufman, 1991, 120~131

[2] Goldberg D E. Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction, Complex Systems 1989(3): 129~152

[3]　Goldberg D E. Genetic Algorithms and Walsh Functions: Part II, Deceptive and its Analysis, Complex Systems, . 1989(3): 153～171

[4]　De Jong, Spears W M. D. F. Gordon: Using Markov Chains to Analysis GAFOs. In: Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, 1994, 115～137

[5]　Goldberg D E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA, Addison－Wisely, 1989

[6]　Kuo T, Hwang S Y. A Genetic Algorithm with Deceptive Selection in Genetic Algorithms. In: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, 1993, 65～69

[7]　Deb K, Goldberg D E. Sufficient Conditions for Deceptive and Easy Binary Functions. Annals of Mathematics and Artificial Intelligence, 1994, 10(4): 385～408

[8]　Suzuki J. A Markov Chain Analysis on Simple Genetic Algorithm. IEEE Trans. SMC, 1995, 25(4): 655～659

[9]　Matsui K, Kosugo Y. An Analysis on Genetic Algorithms Using Markov Processes with Rewards. In: Proceedings of the 1996 IEEE Signal Processing Society Workshop, Sixth in a Series of Workshops Organized by the IEEE Signal Processing Society Neural Networks Technical Committee, 1996, 130～139

[10]　Mahfoud S W. Population Size and Genetic Drift in Fitness Sharing, In: Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, 1994, 185～223

[11]　Blichle T, Thiele L A. Mathematical Analysis of Tournament Selection. In: Proceedings of the 6th International Conference(ICGA95), Eshelman L(ed). Morgan Kaufmann Publishers, 1995

[12]　潘正军，康立山 等. 演化计算. 清华大学出版社，广西科学技术出版社，1998

[13]　史忠植. 高级人工智能. 北京：科学出版社，1998

[14]　张文修，梁怡. 遗传算法的数学基础. 西安：西安交通大学出版社，2000

[15]　刘健勤. 人工生命理论及其应用. 北京：冶金工业出版社，1997

[16]　刘勇，康立山 等. 非数值并行计算(第 2 册)——遗传算法. 北京：科学出版社，1995

第 4 章　遗传算法的改进

自从 1975 年 J. H. Holland 系统地提出遗传算法的完整结构和理论以来，众多学者一直致力于推动遗传算法的发展，对编码方式、控制参数的确定、选择方式和交叉机理等进行了深入的探究，引入了动态策略和自适应策略以改善遗传算法的性能，提出了各种变形的遗传算法（Variants of Canonical Genetic Algorithms，简称 VCGA）。其基本途径概括起来有下面几个方面：

① 改变遗传算法的组成成分或使用技术，如选用优化控制参数、适合问题特性的编码技术等；

② 采用混合遗传算法；

③ 采用动态自适应技术，在进化过程中调整算法控制参数和编码粒度；

④ 采用非标准的遗传操作算子；

⑤ 采用并行遗传算法。

本章将介绍这些方面的典型思路和几种改进的遗传算法。

4.1　分层遗传算法

对于一个问题，首先随机地生成 $N \times n$ 个样本（$N \geqslant 2$，$n \geqslant 2$），然后将它们分成 N 个子种群，每个子种群包含 n 个样本，对每个子种群独立地运行各自的遗传算法，记它们为 GA_i（$i = 1, 2, \cdots, N$）。这 N 个遗传算法最好在设置特性上有较大的差异，这样就可以为将来的高层遗传算法产生更多种类的优良模式。

在每个子种群的遗传算法运行到一定代数后，将 N 个遗传算法的结果种群记录到二维数组 $R[1 \cdots N, 1 \cdots n]$ 中，则 $R[i, j]$（$i = 1 \cdots N$，$j = 1 \cdots n$）表示 GA_i 的结果种群的第 j 个个体。同时，将 N 个结果种群的平均适应度值记录到数组 $A[1 \cdots N]$ 中，$A[i]$ 表示 GA_i 的结果种群平均适应度值。高层遗传算法与普通遗传算法的操作相类似，也可分成如下三个步骤：

1. 选择

基于数组 $A[1 \cdots N]$，即 N 个遗传算法的平均适应度值，对数组 R 代表的结果种群进行选择操作，一些结果种群由于它们的平均适应度值高而被复制，甚至复制多次；另一些结果种群由于它们的种群平均适应度值低而被淘汰。

2. 交叉

如果 $R[i, 1 \cdots n]$ 和 $R[j, 1 \cdots n]$ 被随机地匹配到一起，而且从位置 x 进行交叉（$1 \leqslant i, j \leqslant N$；$1 \leqslant x \leqslant n-1$），则 $R[i, x+1, \cdots n]$ 和 $R[j, x+1, \cdots n]$ 相互交换相应的部分。这一步骤相当于交换 GA_i 和 GA_j 中结果种群的 $n-x$ 个个体。

3. 变异

以很小的概率将少量的随机生成的新个体替换 $R[1 \cdots N, 1 \cdots n]$ 中随机抽取的个体。

至此，高层遗传算法的第一轮运行结束。N 个遗传算法 GA_i（$i = 1, 2, \cdots, N$）可以从相应

于新的 $R[1\cdots N,1\cdots n]$种群继续各自的操作。

在 N 个 GA_i 再次各自运行到一定代数后，再次更新数组 $R[1\cdots N,1\cdots n]$和 $A[1\cdots N]$，并开始高层遗传算法的第二轮运行。如果继续循环操作，直至得到满意的结果。

上述改进的遗传算法，称为分层遗传算法(Hierarchic Genetic Algorithm, HGA)。N 个低层遗传算法中的每一个在经过一段时间后均可以获得位于个体串上的一些特定位置的优良模式。通过高层遗传算法的操作，$GA_i(i=1,2,\cdots,N)$可以获得包含不同种类的优良模式的新个体，从而为它们提供了更加平等的竞争机会。这种改进的遗传算法与并行或分布遗传算法相比，在上一层的个体交换上，它不需要人为地控制应交换什么样的个体，也不需要人为地指定处理器将传送出的个体送往哪一个处理器，或者从哪个处理器接受个体。这样改进的遗传算法不但在每个处理器上运行着遗传算法，同时对各处理器不断生成的新种群进行着高一层的运算和控制。

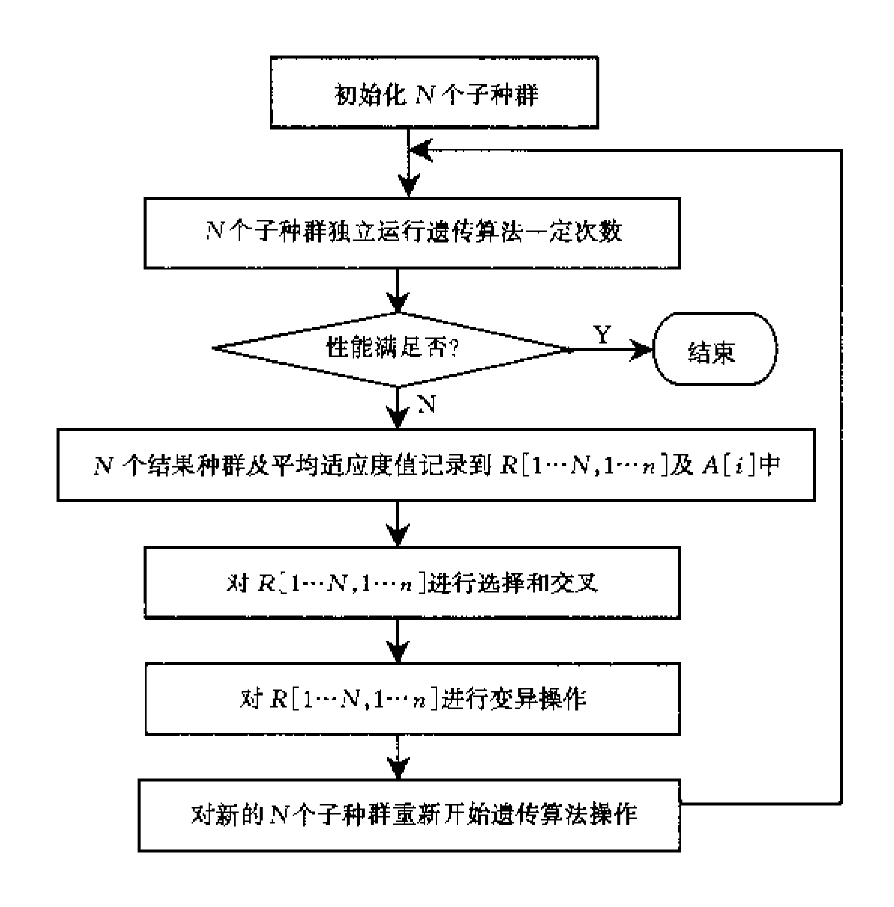


图4.1　分层遗传算法

4.2　CHC 算法

CHC 算法是 Eshelman 于 1991 年提出的一种改进的遗传算法的缩称，第一个 C 代表跨世代精英选择(Cross generational elitist selection)策略，H 代表异物种重组(Heterogeneous recombination)，第二个 C 代表大变异(Cataclysmic mutation)。CHC 算法与基本遗传算法 SGA 不同点在于：SGA 的遗传操作比较单纯，简单地实现并行处理；而 CHC 算法牺牲这种单纯性，换取遗传操作的较好效果，并强调优良个体的保留。下面分别介绍其遗传操作的改进之处。

1. 选择

通常，遗传算法是依据个体的适应度复制个体完成选择操作的，而在CHC算法中，上世代种群与通过新的交叉方法产生的个体群混合起来，从中按一定概率选择较优的个体。这一策略称为跨世代精英选择。其明显的特征表现在：

(1) **健壮性** 由于这一选择策略，即使当交叉操作产生较劣个体偏多时，由于原种群大多数个体残留，不会引起个体的评价值降低。

(2) **遗传多样性保持** 由于大个体群操作，可以更好地保持进化过程中的遗传多样性。

(3) **排序方法** 克服了比例适应度计算的尺度问题。

2. 交叉

CHC算法使用的重组操作是对均匀交叉的一种改进。均匀交叉对父个体位值的各位位置以相同的概率实行交叉操作，这里改进之处是：当两个父个体位值相异的位数为 m 时，从中随机选取 $m/2$ 个位置，实行父个体位值的互换。显然，这样的操作对模式具有很强的破坏性，因此，确定一阀值，当个体间的海明距离(Hamming distance)低于该阀值时，不进行交叉操作。并且，与种群进化收敛的同时，逐渐地减小该阀值。

3. 变异

CHC算法在进化前期不采取变异操作，当种群进化到一定的收敛时期，从优秀个体中选择一部分个体进行初始化。初始化的方法是选择一定比例的基因座，随机地决定它们的位值。这个比例值称为扩散率，一般取0.35。

CHC算法具体描述如下。这里，N 为种群大小，L 为个体长，k 为世代数，d 为海明距离阀值，r 为扩散率。

```
procedure CHC
begin
    k=0;
    d=L/4;
    P(k)初始化;
    P(k)的评价;
      直到满足终止条件终止
    begin
    k=k+1;
    复制P(k-1),与P(k)混合后产生混合种群C(k);
    C(k)中实行交叉,形成新种群C'(k);
    C'(k)的评价;
    从C'(k)和P(k-1)中实行选择;
    if  P(k)==P(k-1)
        d=d-1;
        if  d<0
        begin
        P(k)的一部分个体实行初始化;
        d=r(1-r)L;
      end
    end
```

```
    end
end
procedure 交叉
    begin
      C(k)中个体分别配对;
      begin
      求配对个体间的海明距离;
      if  海明距离>2d
      实行改进的均匀交叉;
      else
        将该配对个体从种群中消除;
      end
    end
end
procedure 选择
begin
    C'(k)按评价值优劣顺序排列;
    while P(k)中的最差者<C'(k)中的最优者
    两者替换;
    end
end
procedure 部分个体初始化
begin
      P(k-1)中挑选较优秀的 N 个个体生成 P(k);
      对 P(k)中 N-1 个个体
      begin
      随机选择 rL 位,其位值随机地决定;
      个体评价;
      end
end
```

4.3　messy GA

根据积木块假设,SGA 中,定义距长的模式容易受到破坏,只有从小积木块的模式中才能最终构成最优解,这对进化模拟而言是十分不利的。为克服这一缺点,Goldberg 等在 1989 年提出了一种变长度染色体遗传算法,该算法在不影响模式定义距的情况下,使优良的模式得以增殖。

在生物进化过程中,其染色体的长度并不是固定不变的,而是随着进化过程也在慢慢地变化。另一方面,在遗传算法的实际应用中,有时为简化描述问题的解,也需要使用不同长度的编码串。例如,用遗传算法对模糊控制器规则库进行优化设计时,事先一般不知道规则数目,这样一个规则库对应一个个体,个体的染色体长度可以描述为变化的。用遗传算法对人工神经网络结构进行优化设计时,如果各层的结点数是未知的,同样,个体的染色体长度也可以描

述为变化的。用遗传算法进化硬件(Evoluable Hardware, EHW)时(详见10.5节),将可编程集成电路结构当作遗传算法中的染色体,PLD的结构可由置载一个称之为结构位串(architecture bit string)的二进制位串来决定,结构位串是由布尔函数的真值表(truth table)描述,由遗传算法找出最佳的硬件结构,若将所有开关集都直接编码染色体,存在电路规模和进化速度问题。实际上只有少数的开关会真正用来决定硬件结构,因此,基因型中存在许多无用染色体,从而导致遗传算法运行时间的增加,用变长度染色体描述这类问题是非常有利的。

messy GA将常规的遗传算法的染色体编码串中各基因座位置及相应的基因值组成一个二元组,把这个二元组按一定顺序排列起来,就组成一个变长度染色体的一种编码方式。一般地它可表示为

$$X^m:(i_1,\ v_1)\ (i_2,\ v_2)\ \cdots\ (i_k,\ v_k)\ \cdots\ (i_n,\ v_n)$$

上述变长度染色体描述形式中,i_k 是所描述的基因在原常规染色体中的基因座编号,v_k 为对应的基因值。对于所需求解的问题,若使用常规遗传算法时的染色体长度固定为 l,各基因值取自集合 V,则有:

$$1 \leqslant i_k \leqslant l \quad (k=1,2,\cdots,n)$$

$$v_k \in V \quad (k=1,2,\cdots,n)$$

例如,常规遗传算法的一个个体的基因型为011101,其染色体长度为6,对于messy GA,该个体可以表示为:X^m:(1,0) (2,1) (3,1) (4,1) (5,0) (6,1),该个体也可表达为:

(2,1) (5,0) (1,0) (3,1) (6,1) (4,1)

在这种算法中,允许染色体的长度可长可短,如:

X^m:(1,1) (2,0) (3,0) (4,1) (5,0) (6,1) (3,1) (1,0)

X^m:(1,1) (3,0) (5,0) (6,1)

前者染色体编码串中出现二元组重复描述,而后者染色体编码串中出现二元组缺失描述。在解码方法上需要一定的规则来约定,对于二元组重复描述的情形,规定取最左边的二元组进行解码;对于二元组缺失描述的情形,规定其基因值取某一预先设定的标准值。因此,上面两个个体的解码对应于常规遗传算法的个体分别为

100101

100001

messy GA由于编码长度可变,遗传操作算子选择具有特殊性,一般选择算子选用锦标赛选择方法(tournament selection),不再使用通用的交叉算子,而代之以切断算子(cut operator)和拼接算子(splice operator)。切断算子是以某一预先指定的概率,在变长度染色体中随机选择一个基因座,使之成为两个个体的基因型;拼接算子是以某一预先指定的概率,将两个个体的基因型连接在一起,使它们合并成一个个体的基因型。这两个算子的处理过程称为并列阶段(juxtapositional phase),并列阶段处理示例如图4.2所示。

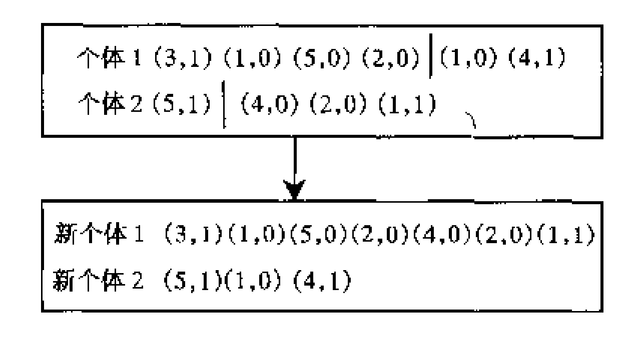


图4.2 messy GA的并列阶段

4.4　自适应遗传算法

遗传算法的参数中交叉概率 P_c 和变异概率 P_m 的选择是影响遗传算法行为和性能的关键所在，直接影响算法的收敛性，P_c 越大，新个体产生的速度就越快。然而，P_c 过大时遗传模式被破坏的可能性也越大，使得具有高适应度的个体结构很快就会被破坏；但是如果 P_c 过小，会使搜索过程缓慢，以至停滞不前。对于变异概率 P_m，如果 P_m 过小，就不易产生新的个体结构；如果 P_m 取值过大，那么遗传算法就变成了纯粹的随机搜索算法。针对不同的优化问题，需要反复实验来确定 P_c 和 P_m，这是一件繁琐的工作，而且很难找到适应于每个问题的最佳值。Srinvivas 等提出一种自适应遗传算法(Adaptive GA, AGA)，P_c 和 P_m 能够随适应度自动改变。当种群各个体适应度趋于一致或者趋于局部最优时，使 P_c 和 P_m 增加，而当群体适应度比较分散时，使 P_c 和 P_m 减少。同时，对于适应值高于群体平均适应值的个体，对应于较低的 P_c 和 P_m，使该解得以保护进入下一代；而低于平均适应值的个体，相对应于较高的 P_c 和 P_m，使该解被淘汰掉。因此，自适应的 P_c 和 P_m 能够提供相对某个解的最佳 P_c 和 P_m。自适应遗传算法在保持群体多样性的同时，保证遗传算法的收敛性。

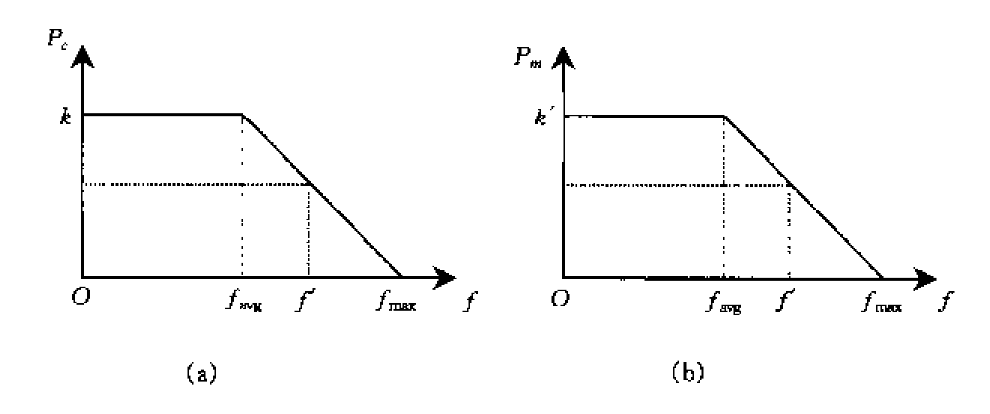


图 4.3　自适应交叉率和变异率

(a) 自适应的交叉概率($k=k_1=k_2$)；(b) 自适应的变异概率($k'=k_3=k_4$)

在自适应遗传算法中，P_c 和 P_m 按如下公式进行自适应调整：

$$P_c=\begin{cases}\dfrac{k_1(f_{max}-f')}{f_{max}-f_{avg}}, & f\geqslant f_{avg}\\ k_2, & f<f_{avg}\end{cases}\tag{4.1}$$

$$P_m=\begin{cases}\dfrac{k_3(f_{max}-f)}{f_{max}-f_{avg}}, & f\geqslant f_{avg}\\ k_4, & f<f_{avg}\end{cases}\tag{4.2}$$

式中，f_{max}——群体中最大的适应度值；

f_{avg}——每代群体的平均适应度值；

f'——要交叉的两个个体中较大的适应度值；

f——要变异个体的适应度值。

这里，只要设定 k_1, k_2, k_3, k_4 取(0,1)区间的值，就可以自适应调整了。

当适应度值低于平均适应度值时，说明该个体是性能不好的个体，对它就采用较大的交叉率和变异率；如果适应度值高于平均适应度值，说明该个体性能优良，对它就根据其适应度值取相应的交叉率和变异率。可以看出，当适应度值越接近最大适应度值时，交叉率和变异率就越小；当等于最大适应度值时，交叉率和变异率的值为零。这种调整方法对于群体处于进化后期比较合适，但对于进化初期不利，因为进化初期群体中的较优的个体几乎处于一种不发生变化的状态，而此时的优良个体不一定是优化的全局最优解，这容易使进化走向局部最优解的可能性增加。为此，可以做进一步的改进，使群体中最大适应度值的个体的交叉率和变异率不为零，分别提高到 P_{c2} 和 P_{m2}，这就相应地提高了群体中表现优良的个体的交叉率和变异率，使得它们不会处于一种近似停滞不前的状态。为了保证每一代的优良个体不被破坏，采用精英选择策略，使它们直接复制到下一代中。

经过上述改进，P_c 和 P_m 计算表达式如下：

$$P_c = \begin{cases} P_{c1} - \dfrac{(P_{c1} - P_{c2})(f' - f_{avg})}{f_{max} - f_{avg}}, & f' \geqslant f_{avg} \\ P_{c1}, & f' < f_{avg} \end{cases} \tag{4.3}$$

$$P_m = \begin{cases} P_{m1} - \dfrac{(P_{m1} - P_{m2})(f_{max} - f)}{f_{max} - f_{avg}}, & f \geqslant f_{avg} \\ P_{m1}, & f < f_{avg} \end{cases} \tag{4.4}$$

上式中 $P_{c1}=0.9$，$P_{c2}=0.6$，$P_{m1}=0.1$，$P_{m2}=0.001$。

4.5 基于小生境技术的遗传算法

生物学上，小生境(niche)是指特定环境中的一种组织功能。在自然界中，往往特征、性状相似的物种相聚在一起，并在同类中交配繁衍后代。在 SGA 中，交配完全是随机的，虽然这种随机化的杂交形式在寻优的初级阶段保持了解的多样性，但在进化的后期，大量个体集中于某一极值点上，它们的后代造成了近亲繁殖。在用遗传算法求解多峰值函数的优化计算时，经常是只能找到个别的几个最优解，甚至往往得到的是局部最优解。我们希望优化算法能够找出全部的最优解，引进小生境的概念，有助于实现这样的目的。

小生境技术就是将每一代个体划分为若干类，每个类中选出若干适应度较大的个体作为一个类的优秀代表组成一个种群，再在种群中以及不同种群之间通过杂交、变异产生新一代个体群，同时采用预选择(preselection)机制或排挤(crowding)机制或分享(sharing)机制完成选择操作。基于这种小生境技术的遗传算法(Niched Genetic Algorithms, NGA)，可以更好地保持解的多样性，同时具有很高的全局寻优能力和收敛速度，特别适合于复杂多峰函数的优化问题。

模拟小生境的方法主要建立在对常规选择操作的改进基础之上。Cavichio 在 1970 年提出了基于预选择机制的选择策略，其基本做法是：当新产生的子代个体的适应度超过其父代个体的适应度时，所产生出的子代个体才能代替其父代个体而遗传到下一代群体中，否则父代个体仍保留在下一代群体中。由于子代个体和父代个体之间编码结构的相似性，所以替换掉的只是一些编码结构相似的个体，故它能够有效地维持群体的多样性，并造就小生境的进化环

境。De Jong在1975年提出了基于排挤机制的选择策略，其基本思想源于在一个有限的生存空间中，各种不同的生物为了能够延续生存，它们之间必须相互竞争各种有限的生存资源。因此，在算法中设置一个排挤因子 CF（一般取 $CF=2$ 或3），由群体中随机地选取的 $1/CF$ 个个体组成排挤成员，然后依据新产生的个体与排挤成员的相似性来排挤一些与排挤成员相类似的个体，个体之间的相似性可用个体编码串之间的海明距离来度量。随着排挤过程的进行，群体中的个体逐渐被分类，从而形成一个个小的生成环境，并维持了群体的多样性。

共享法的选择策略是Goldberg等于1987年提出的。其基本做法是通过个体之间的相似程度的共享函数来调整群体中各个个体的适应度，从而在群体的进化过程中，算法能够依据这个调整后的新适应度来进行选择操作，以维护群体的多样性，创造出小生境的进化环境。共享函数(sharing function)是表示群体中两个个体之间密切关系程度的一个函数，可记为 $S(d_{ij})$，其中 d_{ij} 表示个体 i 与个体 j 之间的某种关系。例如，个体基因型之间的海明距离就可定义为一种共享函数。这里个体之间的密切程度主要体现在个体基因型的相似性或者个体表现型之间的相似性上。当个体之间比较相似时，其共享函数值就比较大，反之，当个体之间不大相似时，其共享函数就比较小。

适应度共享函数的直接目的是将搜索空间的多个不同峰值在地理上区分开来，每一个峰值处接受一定比例数目的个体，比例大小与峰值高度有关。为了实现这样的分布，共享法将个体的目标适应度降低，即适应度值 f_i 除以一个niche计数 m_i 获得共享函数，niche计数 m_i 作为个体邻集密集程度的估计。

$$m_i = \sum_{j \in Pop} Sh[d[i,j]] \tag{4.5}$$

上式中 $d[i,j]$ 是个体 i 和 j 的距离，$Sh[d]$ 是共享函数，它是一个递减函数，$Sh[0]=1$ 和 $Sh[d \geqslant \sigma_{share}]=0$。

下面是一个典型的三角共享函数：

$$Sh(d) = \begin{cases} 1 - \dfrac{d}{\sigma_{share}}, & d \leqslant \sigma_{share} \\ 0, & d > \sigma_{share} \end{cases} \tag{4.6}$$

这里 σ_{share} 是niche半径 r，由使用者自己给定，它是较好峰值之间个体的最小距离。在距离为 σ_{share} 的范围内的个体彼此削减适应度。因为这些个体niche大小相同，因此收敛在一个niche内，避免了整个种群的收敛。当一个niche添满时，其niche计数增大，使共享函数低于其他的niche。Goldberg和Richardson (1987)指出当所有niche的共享函数相等时，就达到一种平衡状态：

$$\frac{f_i}{m_i} = \frac{f_j}{m_j} \quad (i, j \text{ 为 niche 序号}) \tag{4.7}$$

适应度函数共享或多或少独立于使用的选择方法。当共享法与比较流行的竞争选择法结合起来，这种遗传算法会出现混沌现象(Goldberg, 1991)，Goldberg建议将连续变化的共享法与竞争选择法结合起来，niche计数的计算采用当前种群，而不是采用部分添满的下世代种群。

为了定义一个niche，我们采用一种将海明距离测度(基因型差异)与适应度距离(表现型差异)相结合的方法。若 $d_1(x_i, x_j)$ 为任意两个个体 x_i 和 x_j 的海明距离，$d_2(x_i, x_j)$ 是适应度距离，这时共享函数可以定义为

$$S(x_i,x_j)=\begin{cases}1-\dfrac{d_1(x_i,x_j)}{\sigma_1}, & 若\ d_1(x_i,x_j)<\sigma_1,\ d_2(x_i,x_j)\geqslant\sigma_2\\ 1-\dfrac{d_2(x_i,x_j)}{\sigma_2}, & 若\ d_1(x_i,x_j)\geqslant\sigma_1,\ d_2(x_i,x_j)<\sigma_2\\ 1-\dfrac{d_1(x_i,x_j)d_2(x_i,x_j)}{\sigma_1\sigma_2}, & 若\ d_1(x_i,x_j)<\sigma_1,\ d_2(x_i,x_j)<\sigma_2\\ 0, & 否则\end{cases} \tag{4.8}$$

这里 σ_1 和 σ_2 是 niche 半径，即分别为基因型和表现型的作为一个 niche 内个体的最大距离。

个体的适应度函数在共享后变为如下的形式：

$$f'(x_i)=\frac{f(x_i)}{\sum_{j=1}^{M}s(x_i,x_j)} \tag{4.9}$$

4.6 混合遗传算法

我们知道，梯度法、爬山法、模拟退火法等一些优化算法具有很强的局部搜索能力，而另一些含有问题相关的启发知识的启发式算法的运行效率也比较高。如果融合这些优化方法的思想，构成一种新的混合遗传算法(hybrid genetic algorithm)，是提高遗传算法运行效率和求解质量的一个有效手段。目前，混合遗传算法实现方法体现在两个方面，一是引入局部搜索过程，二是增加编码变换操作过程。在构成混合遗传算法时，De Jong 提出下面三个基本原则：

① 尽量采用原有算法的编码；

② 利用原有算法全局搜索的优点；

③ 改进遗传算子。

混合遗传算法的基本构成框架如图 4.4 所示。

4.6.1 遗传算法与最速下降法相结合的混合遗传算法

最速下降法(steepest descent method)也称"梯度法"，是无约束极小化的基本算法。为求目标函数 $f(x)$ $(x\in\mathbf{R}^n)$的极小点，从变量空间的某点 $x^{(k)}$出发，沿 $f(x)$在此点的负梯度 $-\nabla f(x^{(k)})$方向求它的极小点 $x^{(k+1)}$，即求实数 t_k，使

$$f(x^{(k)}-t_k\nabla f(x^{(k)}))=\min_t f(x^{(k)}-t\nabla f(x^{(k)})) \tag{4.10}$$

并令

$$x^{(k+1)}=x^{(k)}-t_k\nabla f(x^{(k)}) \tag{4.11}$$

如此反复进行，得到点列$\{x^{(k)}\}$。在一定条件下，可以证明点列$\{x^{(k)}\}$收敛于 $f(x)$的极小点 x^*，在此法中，初始点是预先选定的。在实际计算时，若给定了终止误差 $\varepsilon>0$，则进行到 $\|\nabla f(x^{(k)})\|<\varepsilon$ 时，计算停止，$x^{(k)}$作为 x^* 的近似值。因为在这一点附近，该点的负梯度方向使函数值下降最快，称之为最速下降法。

最速下降法虽然实现简单，有一阶的收敛速度，但其求得的是局部最优解，而不能保证是全局最优解。针对最速下降法和遗传算法在最优化问题上的不足，综合两种方法各自的优势，产生两者相结合的适合于连续可微函数全局优化问题的混合算法，简称为 HGACSDM(hybrid genetic algorithms combined with steepest descent method)。在这种混合算法中，为加速遗传算

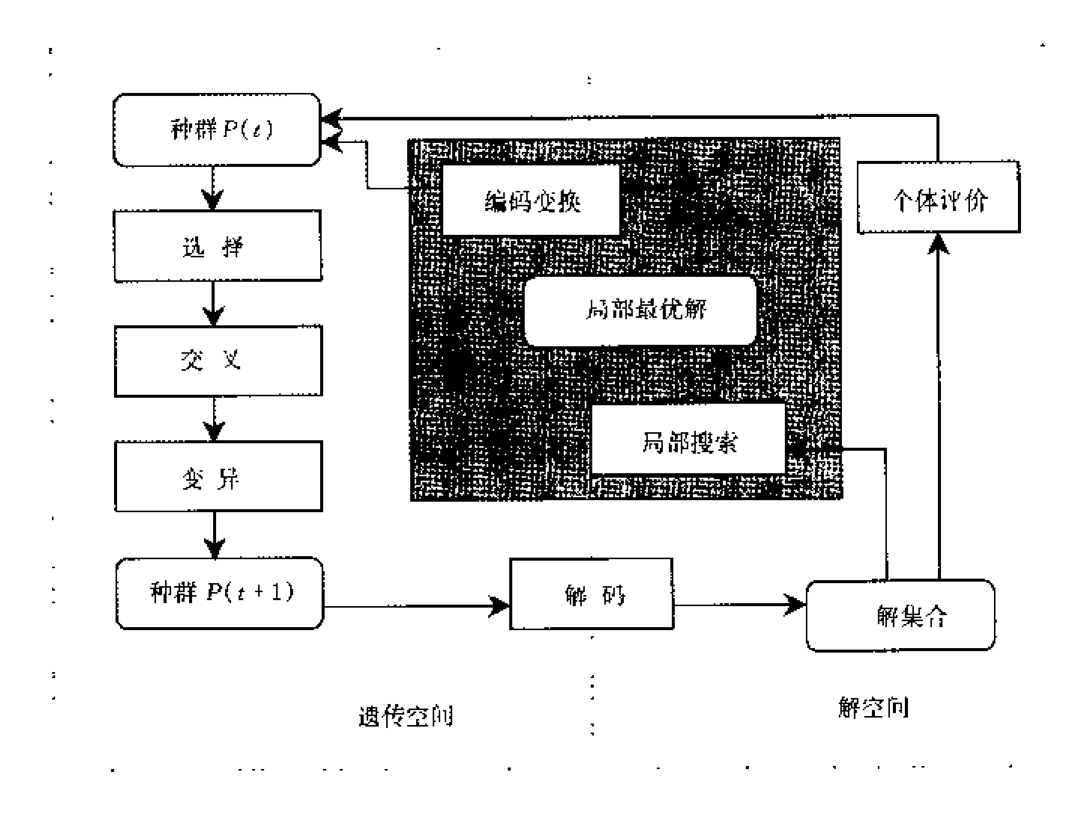


图 4.4　混合遗传算法构成示意图

法在全局优化问题上的收敛性,发挥传统数值优化算法在计算速度与计算精度上的优势,在遗传算法中嵌入一个最速下降算子。该最速下降算子主要进行的是传统最速下降法中的线性搜索运算。在每次繁殖中产生的新的子代,都要以概率 P_s 判断是否需要进行线性搜索运算。由于最速下降法的线性搜索运算是一种能保证迭代产生的点列是函数单调下降的良好的局部极值数值优化算法,因此经最速下降算子的线性搜索运算产生的新的个体继承了其父代的优良品质。因而,可将子代取代父代进入候选群体以待选择新的子代,而不需要父代和子代都进入子代候选群体。

混合算法中的遗传算子——交叉算子、变异算子和选择算子的作用是宏观搜索,处理的是大范围搜索问题,而最速下降算子中的线性搜索过程的作用是极值局部搜索,即微观搜索,处理的是小范围搜索和搜索加速问题。

最速下降算子的线性搜索可采用黄金分割法,概率 P_s 的大小应能保证繁殖过程中对群体中的每个个体都有机会得到一定次数的进行最速下降算子的线性搜索运算。因此,确定最速下降算子概率的 P_s 大小应考虑的因素为:

(1) **所优化函数的线性搜索迭代的收敛性**　若其线性搜索迭代收敛较快,则相应的 P_s 可取小一些。否则,则取大一些。

(2) **预定的进化代数**　若预定的进化代数较大,则相应的 P_s 可取小一些;否则,则取大一些。

4.6.2　遗传算法与模拟退火法相结合的混合遗传算法

下面介绍一种混合遗传算法——模拟退火遗传算法(Simulated Annealing Genetic Algorithm, SAGA)。模拟退火算法是 1982 年 Kirkpatrick 等将固体退火思想引入组合优化领域,

提出了一种解大规模组合优化问题，特别是 NP 完全组合优化的有效近似算法。固体退火过程的物理图像和统计性质是模拟退火算法的物理背景，Metrolpis 接受准则使算法跳离局部最优的“陷阱”，而冷却进度表的合理选择是算法应用的前提。

固体退火是先将固体加热至融化，然后徐徐冷却使之凝固成规整晶体的热力学过程。从统计物理学的观点看，随着温度的降低，物质的能量将逐渐趋近于一个较低的状态，并最终达到某种平衡。

固体温度参数 T，反复进行状态转移过程，新状态的接受概率 $p(x)$服从 Gibbs 分布：

$$p(x) = \frac{1}{z}\exp(-\frac{E(x)}{T}) \tag{4.12}$$

式中，z 为概率正则化系数，$E(x)$为状态 x 的能量。由上式可知，随着温度参数的减小，接受概率也随着减小，即能量函数增大的可能性也逐渐减小，最后系统会收敛于某一能量最小的状态。显然模拟这样的固体退火过程，应用于函数优化中是可行的。

设组合优化问题的一个解 i 及其目标函数分别与固体的微观状态 i 及其能量 E_i 等价。令随着算法进程递减其值的控制参数 t 担当固体退火过程中的温度 T 的角色，则对于控制参数 t 的每一个取值，算法持续进行“产生新解—判断—接受/舍弃”的迭代过程就对应于固体在某一恒定温度下趋于热平衡的过程。从统计物理学获得的 Metrolpis 接受准则应用于确定从当前解 i 到新解 j 转移的概率 P_k：

$$P_k(i \Rightarrow j) = \begin{cases} 1, & \text{当 } f(i) \leqslant f(j) \\ \exp(\dfrac{f(i)-f(j)}{t}), & \text{否则} \end{cases} \tag{4.13}$$

开始时让 t 取较大的值，在进行足够多的状态转移后，缓慢减小 t 的值，如此反复，直至满足某个停止准则时算法终止。因此，模拟退火算法可视为递减控制参数时 Metrolpis 算法的迭代。

下面是模拟退火法的伪代码描述：

```
procedure 模拟退火算法
begin
    S = 初始解 S0;
    T = 初始温度 T0;
    while(某种条件未满足时)
      begin
        while(未达到平衡时)
          begin
            S' = S的邻解;
            Δ = f(S') - f(S);
            Prob = min(1, e^(-Δ/T));
            if  Prob > random(0,1)
            then  S = S';
          end
      update T;
    end
```

Paul L. Stoffa 借鉴模拟退火思想，提出了模拟退火遗传算法(SAGA)，该算法采用如下的

适应度拉伸方法：

$$f_i = \frac{e^{f_i/T}}{\sum_{i=1}^{M} e^{f_i/T}} \tag{4.14}$$

$$T = T_0(0.99^{g-1}) \tag{4.15}$$

式中，f_i 为第 i 个个体的适应度，M 为种群大小，g 为遗传代数，T 为温度，T_0 为初始温度。

遗传算法在运行早期个体差异较大，当采用经典的轮盘赌方式选择，后代产生个数与父个体适应度大小成正比，因此在早期容易使个别好的个体的后代充斥整个种群，造成早熟（premature）；在遗传算法后期，适应度趋向一致，优秀的个体在产生后代时，优势不明显，从而使整个种群进化停滞不前（stalling）。因此对适应度适当地拉伸（scaling or stretching）是必要的。这样在温度高时（遗传算法的前期），适应度相近的个体产生的后代概率相近；而当温度不断下降后，拉伸作用加强，使适应度相近的个体适应度差异放大，从而使得优秀的个体优势更明显。

混合遗传算法中除了上述将遗传算法与最速下降法、模拟退火法结合起来之外，还可以与启发式搜索算法结合起来构成一种新的混合算法；禁忌搜索（Tabu Search, TS）是一种著名的智能启发式搜索算法，由于 TS 具有记忆功能，将之引入到遗传算法的搜索过程中，构造新的重组策略，并把 TS 作为遗传算法的变异算子，这样可以综合两者的优点，并可以克服遗传算法爬山能力差的弱点。

4.7　并行遗传算法

伴随着遗传算法应用的深入开展，并行遗传算法（parallel genetic algorithms, PGA）及其实现的研究也变得十分重要。一般来说，遗传算法中适应度的计算最费时间，再加上需要不断产生新一代，而每一代又有若干个体，所以如何提高遗传算法的运行速度显得尤为突出。由于遗传算法的内在并行机制，其并行处理是很自然的解决途径。

4.7.1　并行遗传算法的实现方案

目前并行遗传算法的实现方案大致可分为三类：

(1) **全局型——主从式模型**（master－slave model）　并行系统分为一个主处理器和若干个从处理器。主处理器监控整个染色体种群，并基于全局统计执行选择操作；各个从处理器接受来自主处理器的个体进行重组交叉和变异，产生新一代个体，并计算适应度，再把计算结果传给主处理器。

1992 年 Abramson 在共享存储的并行计算机上实现了主从式并行遗传算法，用于时间表的优化。主从式方法在适应度评价很费时且远远超过通信时间的情况下才有效，否则通信时间超过计算时间，反而会降低速度。而且这种方法要求有同步机制，这就会导致主进程忙而子进程闲或反之，引起负载不平衡，效率不高。

(2) **独立型——粗粒度模型**（coarse－grained model）　将种群分成若干个子群并分配给各自对应的处理器，每个处理器不仅独立计算适应度，而且独立进行选择，重组交叉和变异操作，还要定期地相互传送适应度最好的个体，从而加快满足终止条件的要求。

粗粒度模型也称孤岛模型（island model），基于粗粒度模型的遗传算法也称为分布式遗传

算法(Distributed Genetic Algorithm, DGA),它是目前应用最广泛的一种并行遗传算法。Pettey 等已证明,综合考虑选择、交叉和变异的效应,粗粒度模型对遗传模式积木块的搜索次数的上限可用指数函数描述,这一结果从理论上表明该模型是有效的。粗粒度模型对并行系统平台要求不高,可以是松散耦合并行系统,特别适合基于 Transputer 的 MIMD 系统,并且效果很好,它主要开发群体之间的并行性。目前在这方面进行的研究有:Petty, Leuze 和 Grefenstette 将一个适应度计算很费时的遗传算法并行化,Tanese 采用四维超立方体结构和固定的迁移间隔,研究并行遗传算法的加速比和解的质量, Conoon, Martin 和 Richards 分析了在并行计算机 Intel 1860 上解图划分问题的多群体遗传算法的性能。

(3) **分散型——细粒度模型(fine-grained model)** 为种群中的每一个个体分配一个处理器,每个处理器进行适应度的计算,而选择、重组交叉和变异的操作仅在与之相邻的一个处理器之间互相传递个体中进行。

细粒度模型也称邻域模型(neighborhood model),适合于连接机、阵列机和 SIMD 系统。目前,这方面的研究有:Manderick 和 Spiessens 基于二维网络拓扑结构开发了一种大规模并行遗传算法,用于研究同类群大小及染色体长度对于算法性能的影响;Muhlenbein 提出了一种异步通信模式并将它应用到复杂组合问题上;Kosak 将群体中个体映射到一个连接机的处理单元,并指出这种方法对网络图设计的有效性。

4.7.2 迁移策略

迁移(migration)是并行遗传算法引入的一个新的算子,它是指在进化过程中子群体间交换个体的过程,一般的迁移方法是将子群体中最好的个体发给其它的子群体,通过迁移可以加快较好个体在群体中的传播,提高收敛速度和解的精度。与单种群相比只需要较少的个体评价计算工作量。因此,即使是采用单一处理器的计算机上以串行方式(伪并行)实现并行算法也能产生较好的结果。因此迁移算子的采用,使并行算法更适合于全局寻优,并且计算量较小。以基于粗粒度模型的并行算法为例,其迁移策略可分为以下两种:

(1) **一传多** 每个处理器对应有若干个相邻处理器,每个处理器产生新一代个体后,都将自己最好的一个个体传送给其所有相邻处理器,并且接受来自相邻处理器的最好的个体,将这些个体与自己的个体同时考虑,淘汰适应度差的个体,其程序流程见图 4.5。

(2) **一传一** 考虑到染色体的多样性,每个处理器都将自己最好的个体仅传给与之相邻的一个处理器,同时增加两个参数:一是 send_rate 决定处理器之间通讯的频率,如 send_rate =3 时表示当遗传代数是 3 的倍数时,各处理器之间相互传送个体;二是 send_best 决定每次传送给最好个体的数目,如 send_best=5 时表示每个处理器把最好的前 5 个个体传给各自的相邻处理器。其程序流程如图 4.6 所示。

一般地,个体迁移的选择方法可以有:

① 均匀随机挑选个体作为迁移对象;

② 按适应度挑选个体作为迁移对象。

对于子种群之间的个体迁移结构也有多种可能性,例如:

① 迁移发生在所有子种群,即发生在完全的网络拓扑,如图 4.7 所示;

② 迁移发生在环状拓扑,如图 4.8 所示;

③ 迁移发生在邻集拓扑,如图 4.9 所示。

最一般化的迁移模型是完全网络拓扑。这里个体在众多的子种群之间相互迁移,移民均

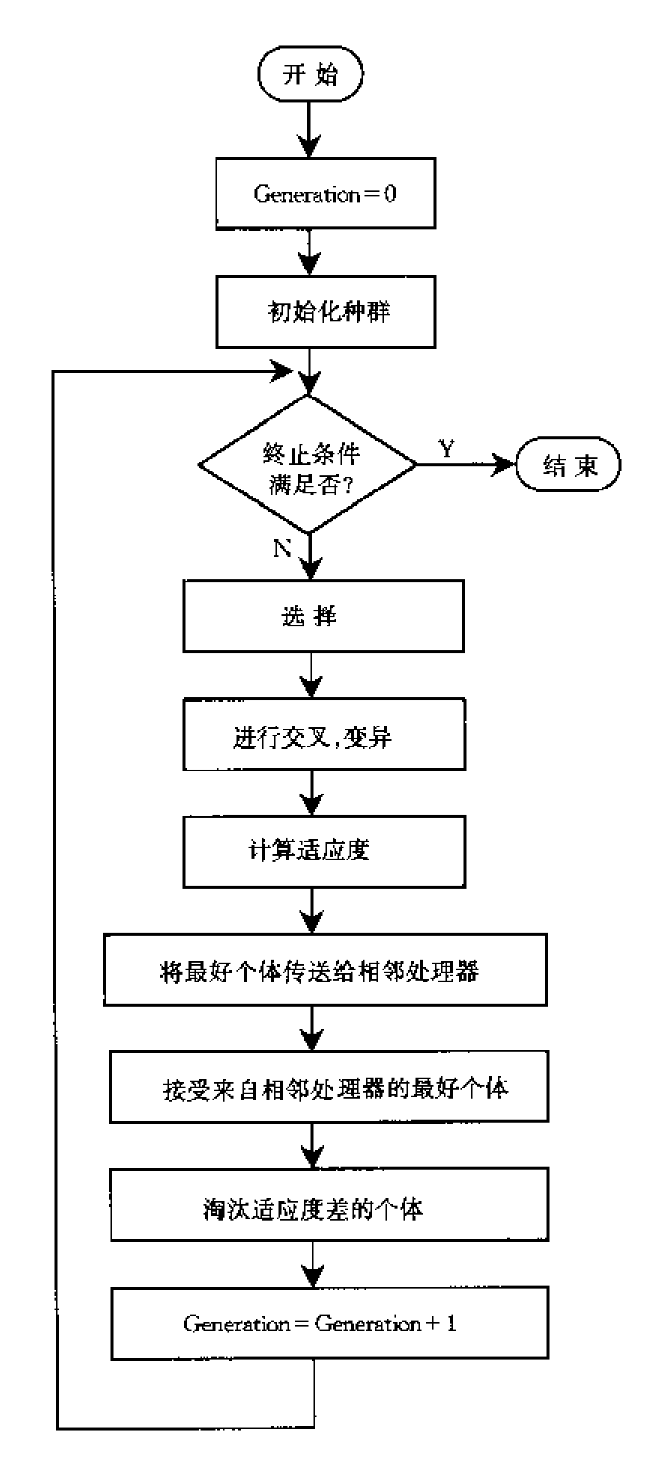


图 4.5　“一传多”程序流程

匀分布在大种群中。图 4.10 给出了四个子种群的这种按适应度选择的无约束迁移。子种群 2,3,4 构造出最佳种群的个体集,随机地从这一个体集中选择一个个体来取代子种群 1 中的最差者。这种过程在每个子种群中循环执行,这样可以保证子种群不会从自身接受个体迁移。

最基本的迁移模型是环状拓扑模型,个体转移只发生在相邻的子种群上。而邻集迁移模型中,迁移只发生在较近的邻集内。

对于多函数的遗传算法,这种多种群的并行算法是很具吸引力的,而且其结果比一般单种

群算法要好。

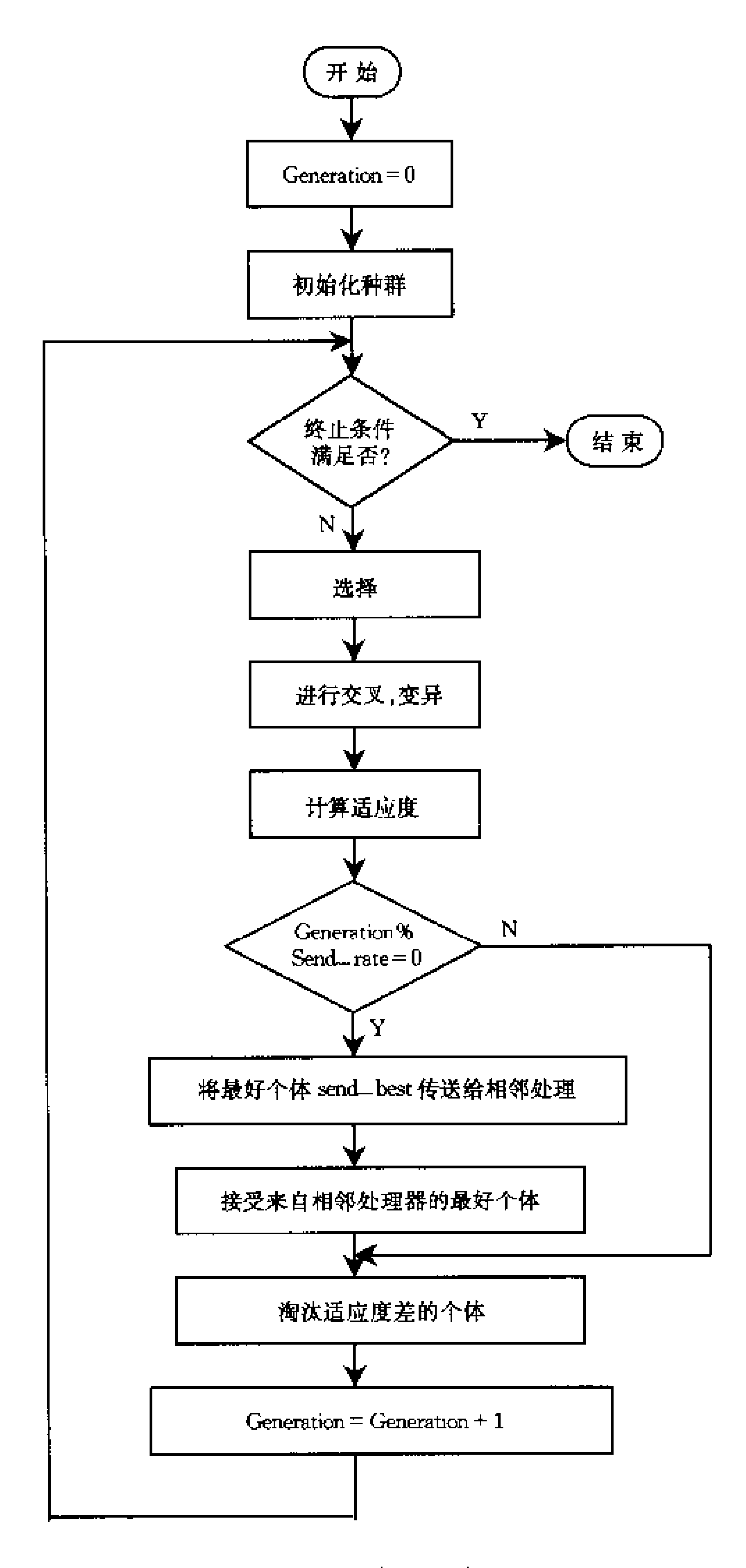


图 4.6 "一传一"程序流程

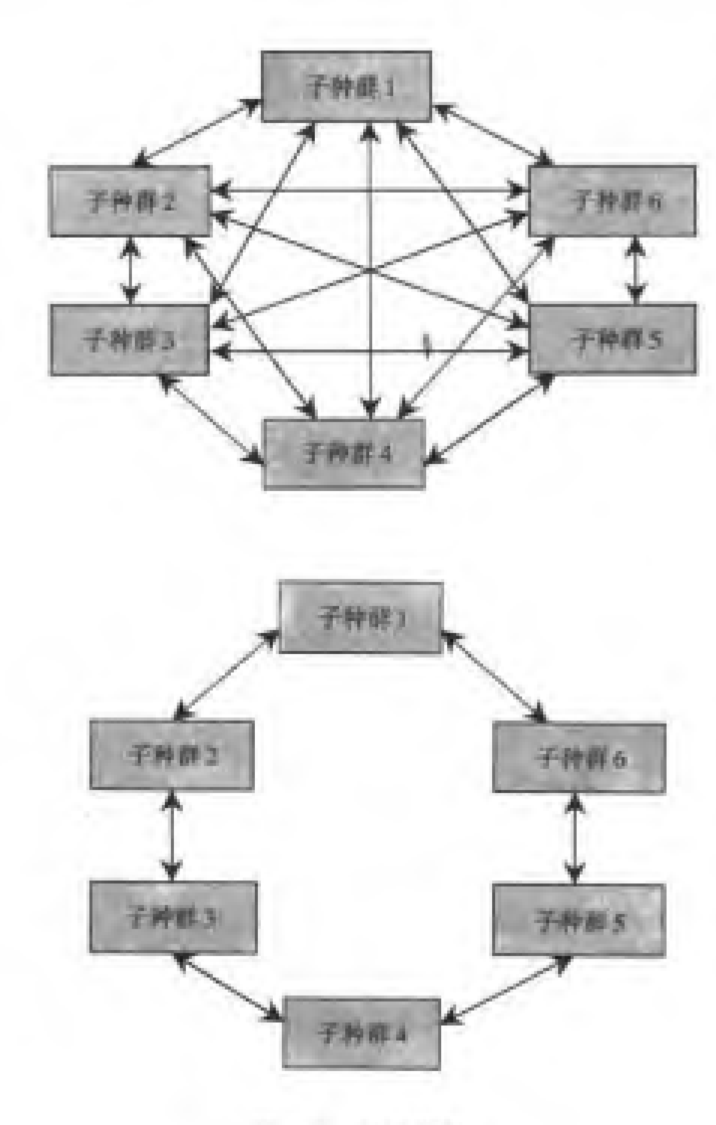


图4.8　环状拓扑

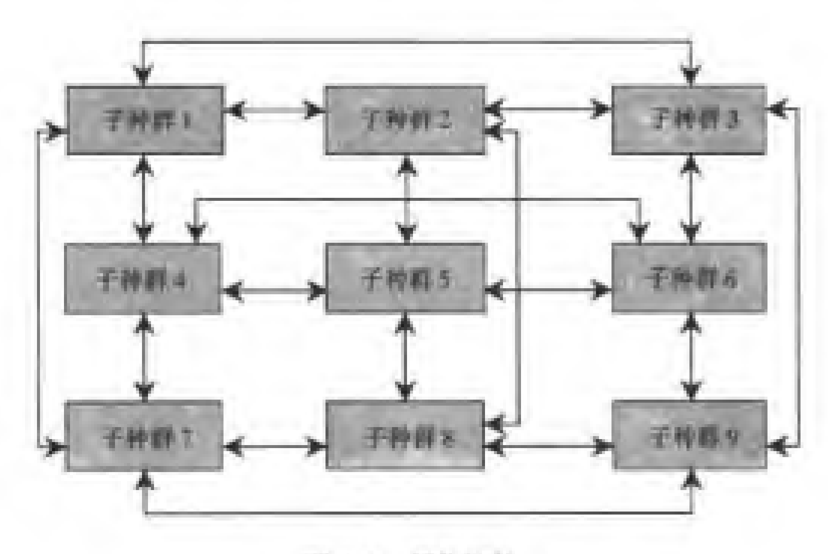


图4.9　邻集拓扑

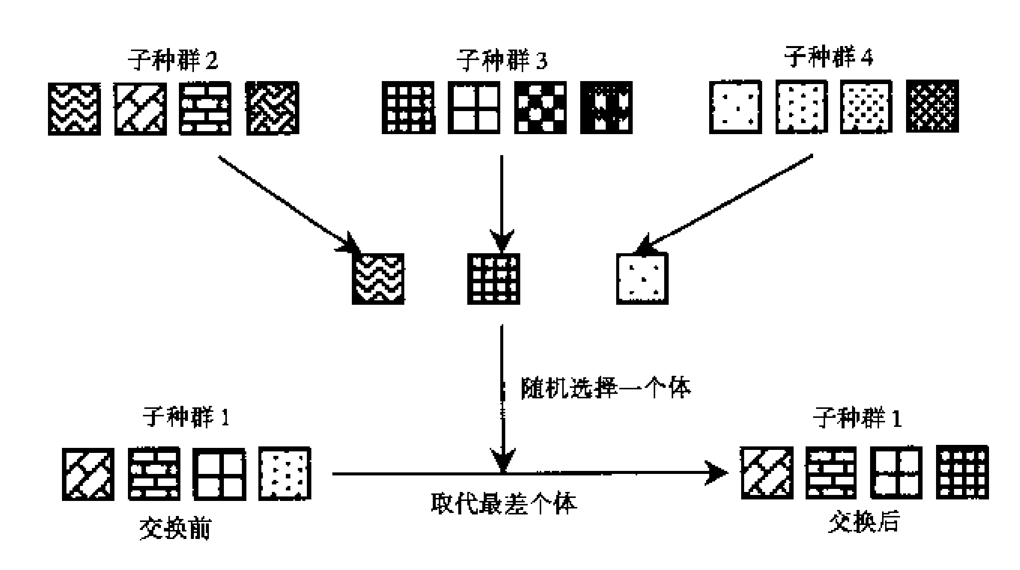


图 4.10 按适应度选择的无约束迁移

4.7.3 并行遗传算法的性能与参数选取关系

为了评价并行算法的性能,人们提出了许多不同的评价指标,其中最重要的一个评价标准是加速比。设 T_1 为某算法在串行计算机上的运行时间,T_p 是该算法在由 p 个处理机所构成的并行机上的运行时间,则此算法在该并行机上的加速比 S_p 定义为:

$$S_p = \frac{T_1}{T_p} \tag{4.16}$$

但对于并行遗传算法,由于搜索的随机性,仅使用加速比这个指标来衡量其性能的优劣程度是比较困难的。比较现实的方法是,设计出一些具有不同几何特性的测试函数,通过它们来测量和统计达到最优点时的平均进化代数和平均计算时间,根据这些测试结果来比较不同的并行遗传算法的优劣。并行遗传算法的性能主要体现在收敛速度和精度两个方面,它们除了与迁移策略有关,还与一些参数选取的合理性密切相关,如遗传代数、群体数目、群体规模、迁移率和迁移间隔。

1. 遗传代数和群体规模

目前大多数研究者采用固定的遗传代数作为算法终止条件,遗传代数越大,求解精度越高,但时间开销越大。如何针对一个具体问题确定一个合理的遗传代数,仍没有一个很好的办法。群体规模是群体个体的数目,群体规模增大有利于解的精度和群体多样性的提高,但同时也增加了求解时间。

2. 迁移率与迁移间隔

将每次被迁移的个体数称为迁移率。迁移率的选取是一个很复杂的问题:由于被迁移者一般均是各子群体中的最优个体,所以迁移率较大,则有利于优良个体在整个群体中的传播和收敛速度的提高,但同时也会增加通信的开销,使加速比下降,也可能导致群体多样性的下降,不利于开发并行遗传算法在多个方向同时进行搜索的特征。应该针对具体问题选取合适的迁移率。

迁移间隔是指相邻两次迁移的时间间隔。迁移间隔小有利于子群体之间的融合,使得优良个体及时传播到所有子群体中,对群体的进化方向可以起到良好的指导作用,有利于提高解的精度和群体的收敛速度。但同时也会明显地增大通信及同步开销,不利于加速比的提高,而

且某些优良个体在群体中的统治地位会产生不利于群体保持多样性的负面影响，使得整个群体类似于串行遗传算法中的随机交配群体(panmixis population)，不利于并行遗传算法发挥其并发搜索多个方向的特性，并有可能使群体进化陷入局部最小点。如果迁移间隔较大，则各子群体之间比较隔绝，其优点是降低了通信开销，提高了加速比，但同时会导致优良个体不能被及时传播，不能充分发挥其导向作用，不利于提高解的精度和收敛速度。

总之，如何获得较好的性能是并行遗传算法中的重要课题。选取合理的参数是十分困难的，这方面目前还没有指导性的实验结论。

参考文献

[1] Whitley D. Modeling Hybrid Genetic Algorithms. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed). Wiley, 1995

[2] Srinivas M, Patnaik L. M. Adaptive Probabilities of Crossover and Mutations in GAs. IEEE Trans. on SMC, 1994, 24(4): 656～667

[3] Goldberg D E, Korb B, Deb K. Messy Genetic Algorithms: Motivation, Analysis and First Results. Complex Systems, 1989(3): 493～530

[4] Fonseca C M, Flemming P J. Multi－Objective Optimization and Multiple Constraint Handling with Evolutionary Algorithms I: A Unified Formulation. Research Report 564, Dept. of Automatic Control and Systems Eng., University of Sheffield, Sheffield, UK, 1995

[5] Fonseca C M, Flemming P J. Multi－Objective Optimization and Multiple Constraint Handling with Evolutionary Algorithms II: Application Example. Research Report 565, Dept. of Automatic Control and Systems Eng., University of Sheffield, Sheffield, UK, 1995

[6] Tamaki, et al. Multi－Criteria Optimization by Genetic Algorithms: A Case of Hot Rolling Process. Technical Report, No. 94－08, Kyoto University, 1994

[7] Surry, et al. A Multi－Objective Approach to Constrained Optimization of Gas Supply Networks. AISB－EC, Sheffield, 1995

[8] Kay C T, Yun L. Multi－Objective Genetic Algorithm Based Time and Frequency Domain Design Unification of Linear Control Systems. Research Report, Center for Systems and Control and Department of Electronics and Electrical Engineering, University of Glasgow, UK, 1995

[9] Cantu－Paz E. A Summary of Research on Parallel Genetic Algorithms. IlliGAL Report No. 95007, 1995

[10] Eshelman L J. The CHC Adaptive Search Algorithm: How to Have Safe Search when Engaging in Non－traditional Genetic Recombination. In: Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, 1991, 265～283

[11] Zitzler E, Thiele L. Multi－Objective Evolutionary Algorithms: A Comparative Case Study And the Strength Pareto Approach. IEEE Transactions of Evolutionary Computation, 1999, 3(4): 257～271

[12] Davis L(ed). Handbook of Genetic Algorithms. Van Nostrand Reinhold, 1991

[13] Nilsson N J. Artificial Intelligence: A New Synthesis. Morgan Kaufman & 机械工业出版社, 1999

[14] Goldberg D E, Rechardson J. Genetic Algorithms with Sharing for Multimodal Optimization. *Proceedings of* the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, 69～76

[15] 周明，孙树栋. 遗传算法原理及其应用. 北京：国防工业出版社，1999

[16] 王宏刚，曾建潮．两级递阶遗传算法．系统工程与电子技术，1998(3)：70～72
[17] 张雪江，朱向阳 等．自适应基因遗传算法及其在知识获取中的应用．系统工程与电子技术，1997(7)：67～72
[18] 李大卫，王梦光．一种改进的混合遗传算法．信息与控制，1997，26(6)：449～454
[19] 徐川育．提高变型标准遗传算法收敛速度的混合法及其推广．信息与控制，1997，26(4)：266～271
[20] 徐金梧，刘纪文．基于小生境技术的遗传算法．模式识别与人工智能，1999，12(1)：104～107
[21] 赵明旺．基于遗传算法和最速下降法的函数优化混合数值算法．系统工程理论与实践，1997(7)：59～64
[22] 段玉倩，贺家李．遗传算法及其改进．电力系统及其自动化学报，1998，10(1)：39～51
[23] 乔建忠，雷为民 等．混合遗传算法研究及其应用．小型微型计算机系统，1998，19(12)：14～19
[24] 王雪梅，王义和．模拟退火法与遗传算法的结合．计算机学报，1997，20(4)：381～384
[25] 郭绚，石晓虹．并行遗传算法性能分析．航空计算技术，1998，28(3)：86～89
[26] 郭绚，郭平，郑守淇．ParaGA：一个并行遗传算法的C++类库．计算机学报，1999，22(6)：591～595
[27] 侯广坤，骆江鹏．一种理想并行遗传算法模型．软件学报，1999，10(5)：557～559
[28] 吴少岩，许卓群．遗传算法中遗传算子的启发式构造策略．计算机学报，1998，21(11)：1003～1008
[29] 向丽，顾培亮．一种基于遗传算法的双层非线性多目标决策方法．系统工程理论方法应用，1999，8(3)：16～21
[30] 李海民，吴成柯．自适应遗传算法及其性能分析．电子学报，1999，27(5)：90～92
[31] 马钧水，刘贵忠，贾玉兰．改进遗传算法的搜索性能的大变异操作．控制理论与应用，1998，15(3)：404～407

第5章 进化计算初步

近30年来，人们从不同的角度对生物系统及其行为特征进行了模拟，产生了一些对现代科技发展有重大影响的新兴学科。例如，对人类模糊思维方式的模拟产生了模糊集合理论；对动物脑神经的模拟产生了人工神经网络理论；对自然界中动、植物免疫机理的模拟产生了免疫算法；而对自然界中生物进化机制的模拟就产生了进化计算(Evolutionary Computation, EC)理论。进化计算最初具有三大分支：遗传算法(Genetic Algorithm, GA)、进化规划(Evolutionary Programming, EP)和进化策略(Evolution Strategy, ES)。90年代初，在遗传算法的基础上又形成了一个分支：遗传程序设计(Genetic Programming, GP)。虽然这几个分支在算法实现方面有一些差别，并且这些差别正逐渐缩小，它们一个共同的特点是借助生物进化的思想和原理来解决实际问题。作为进化计算理论体系的中心——遗传算法，其理论和方法不断地完善具有普遍的意义。本章将讨论进化计算的理论框架及其分支的构成技术，重点介绍遗传程序设计技术。

5.1 进化计算理论的基本框架

进化计算是指以进化原理为仿真依据，在计算机上实现的具有进化机制的算法和程序。目前，进化计算侧重于算法的研究，因此有时也称之为进化算法(evolutionary algorithms, EA)，若由性质来区分，现有的进化算法可细分如下：

① 最具代表性、最基本的遗传算法(genetic algorithm)(第2章)；

② 侧重于数值分析的进化策略(evolution strategy)(5.2节)；

③ 介于数值分析与人工智能间的进化规划(evolutionary programming)(5.3节)；

④ 偏向进化的自组织和系统动力学特性的进化动力学(evolutionary dynamics)；

⑤ 偏向以程式表现人工智能行为的遗传程序设计(genetic programming)(5.4节)；

⑥ 适应动态环境学习的分类系统(classifier system)(第8章)；

⑦ 用以观察复杂系统交互的各种生态模拟系统(echo system and etc.)；

⑧ 研究人工生命(artificial life)的元胞自动机(cellular automata)；

⑨ 模拟蚂蚁群体行为的蚁元系统(ant system)(10.3节)。

进化计算作为一个新兴学科，其框架结构可以用图5.1表示。其与相关学科的交叉应用关系如图5.2所示。

基因型与表现型之间的关系是进化计算中的一个基本关系，在其复杂的非线性关系中“一因多果”和“一果多因”是突出的两个特点。表现型的变化是对象遗传结构和当时环境条件交互作用所致的结果，非线性效果很明显，甚至相差很大的遗传结构可能会导致类似的行为。这

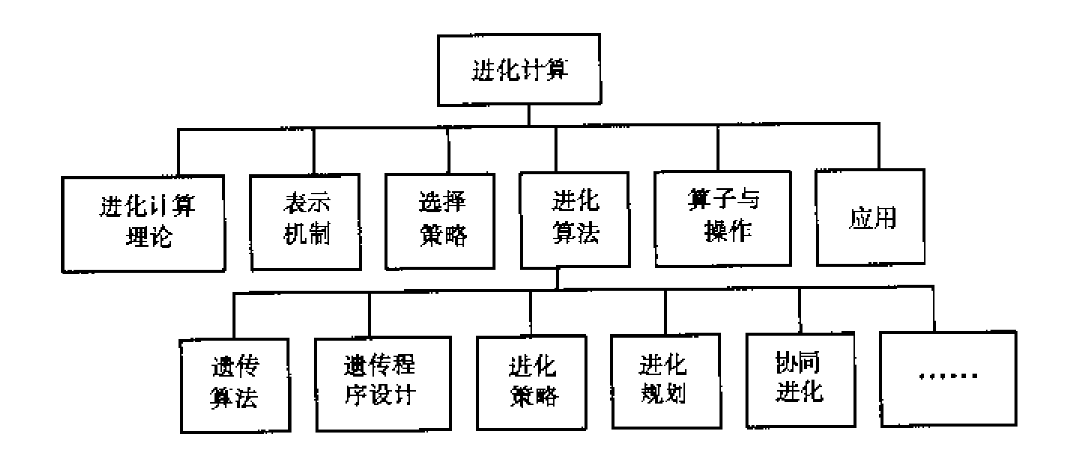


图 5.1 进化计算理论框架结构

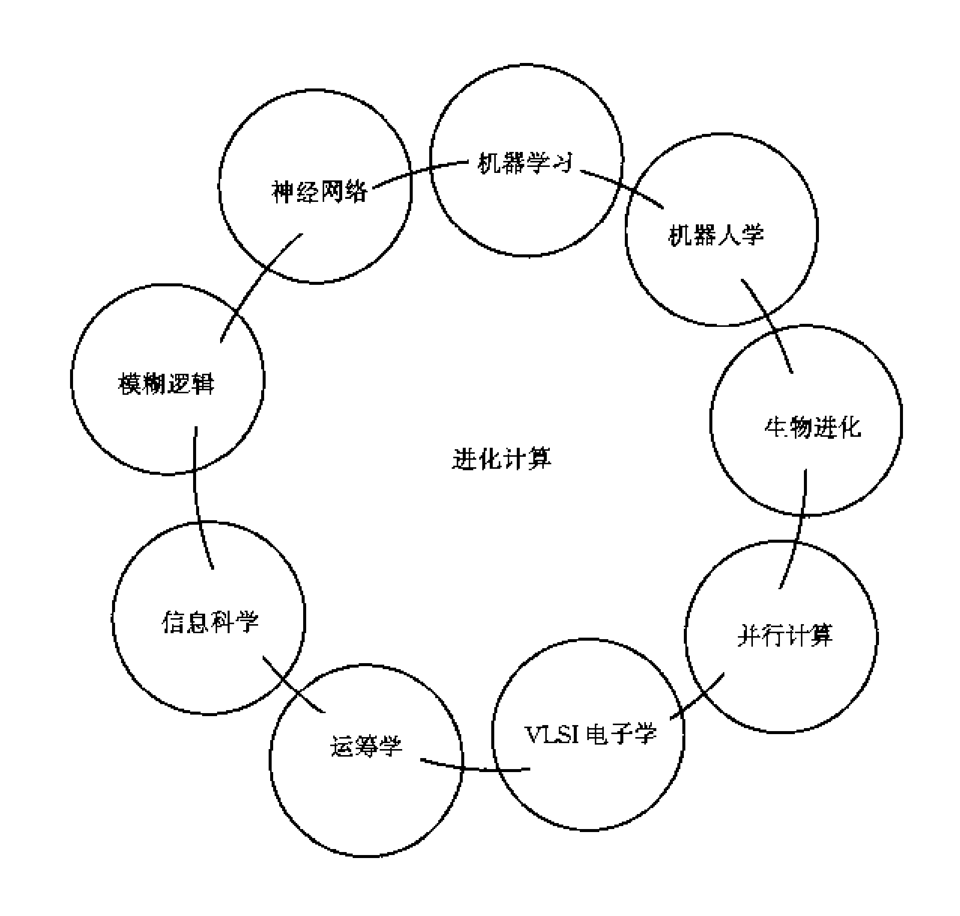


图 5.2 进化计算与相关学科的交叉应用关系

样在研究基因型——表现型相互关系及其在进化过程中的规律时就必须充分利用非线性系统工具和随机过程的统计测度理论,动力学机制也是必须加以研究的动态属性。适应度状态图描述了适应度与基因型间的关系,自适应状态图则勾画了适应度与表现型之间的联系状况。在进化计算中算子占有重要的地位,相应的操作也应反映动态的机制。因此,进化计算体系可归纳为:

进化计算 = 进化算子 + 进化操作 + 选择策略 + 进化计算理论

这里进化算子可以有多种形式,进化策略可容纳较为复杂的非线性动力学机制。进化计

算作为完整的体系，应包括最优性、统计分析、选择策略和相应判据等内容。

5.2 进化策略

20世纪60年代初，柏林工业大学的学生 I. Rechenberg 和 H. P. Schwefel 等在进行风洞实验时，由于设计中描述物体形状的参数难以用传统方法进行优化，因而利用生物变异的思想来随机改变参数值，获得了较好的结果。随后，他们对这种方法进行了深入的研究和发展，形成了进化计算的另一个分支——进化策略(Evolutionary Strategy, ES)。

5.2.1 (1+1)-ES

早期进化策略的种群中只包含一个个体，而且只使用变异操作，所使用的变异算子是基于正态分布的变异操作。在每一进化代，变异后的个体与其父体进行比较，选择两者之优。这种进化策略称为(1+1)-ES或二元ES。

设初期个体为 $x(0)\in \mathbf{R}^n$，它可以代表问题解空间的一个向量。第 k 代($k=1,2,\cdots$)的个体可以作为其父代即第 $k-1$ 代的子个体：

$$x'(k) = x(k-1) + N_n(0,\sigma^2(k)) \tag{5.1}$$

其中 $N_n(0,\sigma^2(k))$ 为独立正态随机向量。

子个体与亲代个体进行比较选择最优者作为当代个体，对于最小化问题，有：

$$x(k) = \begin{cases} x'(k), & f(x'(k)) < f(x(k-1)) \\ x(k-1), & \text{其他} \end{cases} \tag{5.2}$$

(1+1)-ES存在很多弊端，如有时收敛不到全局最优解、效率较低等。它的改进即增加种群内个体的数量，从而演化为($\mu+\lambda$)-ES。

5.2.2 (μ+λ)-ES, (μ, λ)-ES

Schwefel 于1977年提出了($\mu+\lambda$)-ES和(μ,λ)-ES，其基本做法是：种群内含有 μ 个个体，随机选取一个个体进行变异，然后取代种群中最差的个体。($\mu+1$)-ES与(1+1)-ES的相似之处是，每次只产生一个后代。为了进一步提高效率，又发展成为($\mu+\lambda$)-ES和(μ, λ)-ES，并引进了交叉操作。

假设组成进化群体的每一个个体有两部分组成，其中一部分是可以取连续值的向量，另一部分是一个微小的变动量。这个变动量由步长 $\sigma\in\mathbf{R}^n$（正态分布的标准差）和回转角 $\alpha\in\mathbf{R}^{n(n-1)/2}$（正态分布的协方差）组成的，它们可用来调整对个体进行变异操作时变异量的大小与方向。因此群体中的每一个个体 X 可表示为：

$$X = \{x,\sigma,\alpha\}$$

这样，对于每一个个体，它就是空间 I 中的一个点，即：

$$X \in I = \mathbf{R}^{n+n(n-1)/2+n} = \mathbf{R}^{n+n(n+1)/2}$$

一般情况下可以不考虑回转角这个参数，则有：

$$X \in I = \mathbf{R}^{2n}$$

此时，群体中的每一个个体 X 可表示为：

$$X = \{x,\sigma\}$$

每个个体的适应度等于所对应的目标函数，而不再对所要求的目标函数进行任何变换处

理,这主要是由于进化策略中的选择运算是按照一种确定的方式进行的。每次都是从当前群体中选择出一个或几个适应度最高的个体保留到下一代群体中,这里只有个体适应度之间的大小关系比较,而无算术运算,所以对个体适应度的正负无特别的要求。

在进化策略中,变异操作是产生新个体的一种最主要的方法。假设群体的个体 $X=\{x,\sigma\}$ 经过变异运算后得到一个新的个体 $X'=\{x',\sigma'\}$,则新个体的组成元素是:

$$\sigma'_i=\sigma_i\exp[\tau\cdot N(0,1)+\tau'\cdot N_i(0,1)]\quad(i=1,2,\cdots,n)\tag{5.3}$$

$$x'_i=x_i+N(0,\sigma'_i)\quad(i=1,2,\cdots,n)\tag{5.4}$$

式中,$N(0,1)$是均值为0、方差为1的正态分布随机变量,τ 和 τ' 是算子集参数,分别表示变异运算时整体步长和个体长。

在进化策略中,交叉操作只是一种辅助的搜索运算。假设 $X_a=\{x_a,\sigma_a\}$, $X_b=\{x_b,\sigma_b\}$ 为群体中随机配对的两个个体,则对这两个个体进行交叉操作产生出一个新个体 $X'_a=\{x'_a,\sigma'_a\}$,其中的元素 x'_a 可按下列方式之一来确定:

① 无交叉
$$x'_a=x_a\tag{5.5}$$

② 直接交叉
$$x'_a=\begin{cases}x_a, & 若\ random(0,1)=0\\ x_b, & 若\ fandom(0,1)=1\end{cases}\tag{5.6}$$

③ 加权平均交叉
$$x'_a=x_a+\theta(x_b-x_a)\tag{5.7}$$

式中,θ 为[0,1]范围内的随机分布的随机数。

在进化策略中,选择操作是按照一种确定的方式进行的。目前进化策略中所适应的选择方法主要有如下两类:

一类根据种群内的 μ 个个体产生 λ 个个体(用变异和交叉),然后将这 $\mu+\lambda$ 个个体进行比较,从中选取 μ 个最优者,将它们保留到子代群体中。这类选择方法记为$(\mu+\lambda)$选择。

另一类是在新产生的 $\lambda(\lambda\geqslant\mu\geqslant1)$个个体中选取 μ 个最优者,将它们保留到子代群体中。这类选择方法记为(μ,λ)选择。

这两类选择方法对应的进化策略分别被称为$(\mu+\lambda)-$ES和$(\mu,\lambda)-$ES。其一般形式描述为:

```
begin
  Generation: = 0
  初始化:设父代个体数 μ,并进行个体初始化;
  适应度计算;
  while(终止条件不满足)
    变异:通过对父代个体加高斯变异产生 λ 个个体;
    适应度再计算:计算 λ 个个体的适应度;
    选择:if(采用((μ+λ)-ES))
    then:μ 个父个体与 λ 个子个体共同竞争,选择适应度高的 μ 个个体进入新一代的群体;
    else:仅 λ 个个体竞争,选择适应度高的 μ 个个体进入新一代的群体;
  Generation: = Generation + 1;
  end
end
```

将两种进化策略进行比较,$(\mu+\lambda)-$ES较好地继承了父代的优良特性,收敛性好,但易

于陷入局部最小。(μ,λ)-ES 则易于跳出局部最小，但由于放弃了上一代的结果，所以收敛较慢。

5.2.3　进化策略的主要特点

由进化策略的主要构成技术可知，与遗传算法相比，进化策略具有下面的主要特点：

① 遗传算法要将原问题的解空间映射到位串空间中，然后再施行遗传操作，它强调个体基因结构的变化对其适应度的影响；而进化策略是直接在解空间上进行操作，它强调进化过程中从父体到后代行为的自适应性和多样性。从搜索空间的角度来说，进化策略强调演化过程中搜索方向和步长的自适应调节。

② 进化策略中各个个体的适应度直接取自它所对应的目标函数，选择运算是按照确定的方式来进行的，每次从群体中选取最好的几个个体，将它们保留到下一代的群体中。个体的变异运算是进化策略中的主要搜索技术，而个体之间的交叉运算只是作为辅助搜索技术。这与遗传算法是不同的。

进化策略一般只适合求解数值优化问题。近年来，遗传算法也采用十进制编码(或称浮点数编码)技术来求解数值优化问题。ES 与 GA 已呈相互渗透之势。

进化策略的应用实例参见 9.2 节。

5.3　进化规划

进化规划(Evolutionary Programming, EP)的方法最初是由 L. J. Fogel 等在 20 世纪 60 年代提出的。他们在人工智能的研究中发现，智能行为就是具有预测其所处环境的状态、并按给定目标做出适当响应的能力。在研究中，他们将模拟环境描述成由有限字符集中的符号组成的序列。于是问题转化为，怎样根据当前观察到的符号序列做出响应，以获得最大收益。这里，收益按环境中将要出现的下一个符号及预先定义好的效益目标来确定。进化规划中常用有限自动机(Finite State Machine, FSM)来表示这样的策略。这样，问题就变成如何设计一个有效的 FSM。L. J. Fogel 等借用进化的思想对一群 FSM 进行进化，以获得较好的 FSM。他们将此方法应用到数据诊断、模式识别和分类及控制系统的设计等问题中，取得了较好的结果。20 世纪 90 年代，D. B. Fogel 借助进化策略方法对进化规划进行了发展，并用于数值优化及神经网络的训练等问题中获得成功。这样，进化规划就演变为一种优化搜索算法，并在很多实际领域得到应用。

5.3.1　进化规划基本原理与方法

进化规划的基本思想也是源于对自然界中生物进化过程的一种模仿。其构成技术与进化策略的构成技术相类似。在进化规划中搜索空间是一个 n 维空间，与此相对应，搜索点就是一个 n 维向量 $x\in \mathbf{R}^n$。算法中组成进化群体的每一个个体 X 就直接用这个 n 维向量表示，即：

$$X = x \in \mathbf{R}^n$$

个体的适应度 $F(X)$ 是由它所对应的目标函数 $f(x)$ 通过某种比例变换而得到的，这种比例变换是为了既保证各个个体的适应度总取正值，又维持各个个体之间的竞争关系，即个体的适应度由下式确定：

$$F(X) = \delta[f(x)] \tag{5.8}$$

式中 δ 为某种比例变换函数。

与遗传算法和进化策略不同的是，进化规划是从整体的角度出发来模拟生物进化过程的，它着眼于整个群体的进化，强调的是“物种的进化过程”。所以在进化规划中不使用个体交叉之类的遗传操作，个体的变异操作是唯一的一种最优个体搜索方法，这也是进化规划的独特之处。

在进化规划中，变异操作适应高斯变异算子，假设群体中某一个个体 $X = \{x_1, x_2, \cdots, x_n\}$，经过变异运算后得到一个新的个体 $X' = \{x_1', x_2', \cdots, x_n'\}$，则新的个体的组成元素是：

$$x_i' = x_i + \sigma_i \cdot N_i(0,1) \quad (i = 1,2,\cdots,n) \tag{5.9}$$

式中，

$$\sigma_i = \sqrt{\beta_i \cdot F(x) + \gamma_i} \quad (i = 1,2,\cdots,n) \tag{5.10}$$

其中 $N_i(0,1)$是均值为 0、方差为 1 的独立正态随机变量，系数 β_i，γ_i 是特定的参数，一般取 $\beta_i = 1$，$\gamma_i = 0$。

在进化规划中，选择操作是按照一种随机竞争的方式进行的，其基本做法类似于遗传算法中的排序选择。首先将 μ 个父代个体 $P(t)$和经过一次变异运算后所产生的 μ 个子代个体 $P'(t)$合并在一起，组成一共含有 2μ 个个体的集合$\{P(t) \cup P'(t)\}$，对这个集合中的每一个个体 $X_k \in \{P(t) \cup P'(t)\}$，再从这个集合中随机选择 q 个个体，比较这 q 个个体与个体X_k 之间的适应度大小，以其中适应度比 X_k 的适应度高的个体的数目作为个体 X_k 的得分 $W_k(k = 1,2,\cdots,2\mu)$，最后对这 2μ 个按降序排列，选择前 μ 个个体组成进化过程中的新一代群体 $P(t+1)$。由上述选择过程可以获知，每代群体中最好的个体总被赋予了最高的得分，从而这个最好的个体能够被保留到下一代群体中。

5.3.2 进化规划的主要特点

与遗传算法和进化策略相比，进化规划具有下面几个特点：

① 进化规划对生物进化过程的模拟主要着眼于物种的进化过程，算法中不使用个体交叉算子，而主要依靠变异操作。

② 进化规划中的选择运算着重于群体中各个体之间的竞争选择，当竞争数目 q 较大时，这种选择也就类似于进化策略中的确定选择过程。

③ 进化规划直接以问题的可行解作为个体的表现形式，无需再对个体进行编码处理，也无需再考虑随机扰动因素对个体的影响，因而便于应用。

④ 进化规划以 n 维实数空间上的优化问题为主要处理对象。

进化规划方法的应用实例参见 9.1 节。

5.4 遗传程序设计

自计算机出现以来，计算机科学的一个重要目标就是让计算机自动进行程序设计，即只要明确地告诉计算机要解决的问题，而不需要告诉它如何去做。目前处于发展中的机器学习、自组织自适应系统方法、神经网络方法等，试图寻求一些无需明晰表达的求解程序来模拟复杂系统的结构和功能。遗传程序设计(Genetic Programming, GP)引入自定义函数及动态程序复用

方法,为这一问题的解决提供了另一种可能的回答。遗传程序设计的思想是 Stanford 大学的 J.R.Koza 在90年代初提出的,与遗传算法的基本思想相似,但在求解程序的自适应进化模拟中发展了结构上的复杂性。他于1992年出版了专著《遗传程序设计:论采用自然选择方法的计算机编程》,1994年又出版了第二卷《遗传程序设计Ⅱ:再论使用程序的自动发现》。他的代表性论文有《计算机程序的遗传进化和共同进化》、《人工生命:自动复制和改进计算机程序的突现性》,Koza 所讲的"程序"术语的背景来源于 LISP 语言编写遗传程序的形式特性。LISP 语言中的S表达式突出表达了树结构的符号描述机制,LISP 这种人工智能语言本身是由符号表的方式的S表达式结构组成的,所以这里的"程序"和"语句"具有某种同一性。"程序"的大小、形式和结构不是预先能够确定的,而是在世代演化中动态地变化。

由于遗传程序设计采用一种更自然的表示方式,因而应用领域非常广,在解决人工智能、机器学习、控制及分子生物学等领域中的问题效果尤其显著。尽管目前遗传程序设计的水平不足以产生出完善的程序(特别是 LISP 语言以外的计算机程序语言形式),其巨大的应用潜力已经受到许多学者特别的关注。日本 ATR 研究中心的 H.deGaris 甚至提出,遗传程序设计不仅可以进化计算机程序,而且可以进化任何复杂系统。他目前正在研究的 CAM-Brain 计划,便是基于这种思想。该计划的目标是要制造具有人脑部分功能的人工脑(Artificial Brain)。

5.4.1 遗传程序设计的原理与方法

从广义问题求解的观点来看,人工智能、符号推理和机器学习中的许多方法可被视为是旨在找出根据样本集就能产生合理信号集方案的某种计算机编程过程。遗传程序设计是在可能空间中寻找合适的计算机程序的自适应搜索算法。其搜索过程从本质上属于随机搜索算法,但其空间遍历性比传统的启发式搜索要好,遗传操作使得路径可随机跳跃至不同的子空间,从而在全局空间中以若干搜索集的并集方式从时序过程方面逼近全集。因此遗传算法是一种逼近求解方法。

GP是在自然选择的基础上,在计算机时代才能实现的一种解题方法。其基本思想是:随机产生一个适合于给定环境的初始群体,即问题的搜索空间。与遗传算法类似,构成群体的个体都有一个适应度值,依据达尔文适者生存原则,用遗传算子处理得到高适应度的个体,产生下一代的群体,如此进化下去,给定问题的解或近似解将在某一代上出现。

1. 遗传程序设计的结构

作为GP的重要特点之一,组成群体的个体采用一种动态的树状结构。树的结点由终结点(terminals)、原始函数(primitive functions)与运算符组成。其中终结点也称为叶结点,它是将问题分级划分为子问题后最基本的解的成分。这种树状的层与结点均是可变化的。终结点是问题的原始变量,根结点和中间结点统称为内部结点,它们则是组合这些原始变量的函数。它们很类似于 LISP 语言中的S表达式。这样,每个分层结构对应问题的一个可能解,也可以理解为求解该问题的一个计算机程序。图5.3为一个函数计算程序的树状结构例子。

自动定义函数(Automatically Defined Functions, ADF)是GP的一个重要概念。自动定义函数是GP在运行过程中,根据定义规则由函数叶结点与运算符组合而成的树状结构,其生成方法与GP生成个体的方法一样。它试图将问题分级,求出了问题的解。自动定义函数可以在个体中反复使用,如果在GP运行中,某个自动定义函数有价值,它肯定反映了问题的某种规律性东西,此函数将被保留下来,否则将被淘汰。很明显,这种在运行中自动产生的、能够自

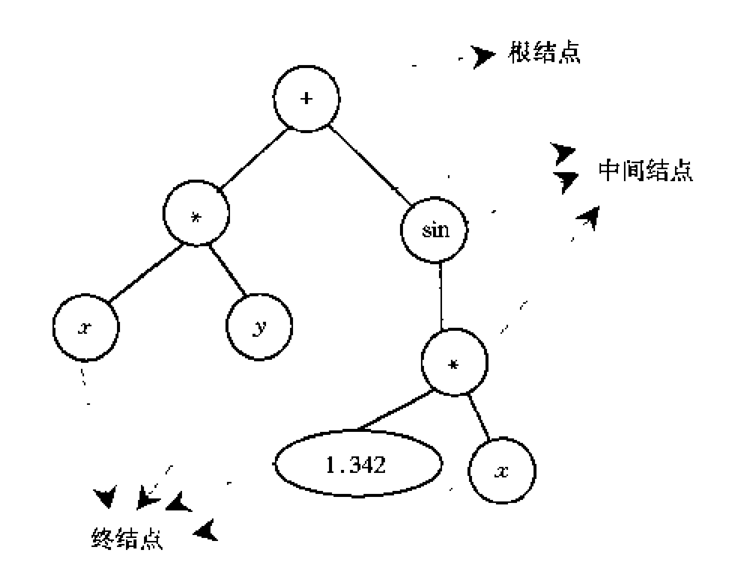


图 5.3 计算式 $(x*y)+(\sin(1.342*x))$ 的树结构示意图

动提取问题某些特性的函数可以说具有学习的能力，它是 GP 的一个有力工具。

利用自动定义函数，GP 将复杂问题分解为小的简单子问题，然后将子问题的解自动装配为整个问题的解。借助自动定义函数，GP 可以发现问题环境的某些规律性，如相似性(simularities)、规律性(regularities)、对称性(symmetrices)、模式(patterns)和模块性(modularities)等，自动定义函数使 GP 的解题能力具有自然增长性、高效性与健壮性。

2. 遗传程序设计的算法

GP 的算法设计需要预先确定以下五个问题：

(1) 定义终点集 T

终点集为表示问题环境与结果的最基本的元素，依问题的不同，元素的含义不同。

(2) 定义初始函数集 F

初始函数可以是算术运算(+, −, *, /)、数学函数(sin, cos, exp 和 log)、布尔运算(AND, OR, NOT)、条件运算(If-then-else)、迭代算子(Do-until)、递归函数、程序运算等，函数可以返回某些值或者仅执行某种操作。不同函数的自变量个数不同，且函数应具有终结点集封闭性，即函数集 F 中的任何函数的返回值仍属于终点集。对于出现无穷大、分母为 0、无穷次循环等非法运算，应预先给定判断和处理。此外，初始函数依据有效性的原则从上述类型的函数中遴选，遴选出来的初始函数集、终点集以及预先确定的树结构的最大深度，应能够有效地表达问题的解。

(3) 适应度值的评价方法

适应度的评价驱动进化过程，适应度的值给出了问题环境下群体中每个个体的好坏程度，它应有能力对所遇到的任何一代群体中的个体进行估计。

(4) 确定运行控制量

控制 GP 一次运行的基本参量为：群体大小 M，运行最大代数 G，另外还可以有第二级参数。

(5) 终止运行的标准

一次 GP 的应用通常是 GP 程序独立运行的结果的综合，所以需要确定 GP 应用的终止标

准,决定何时最终停止运行GP,同时还需确定选择最终结果的方法,以获取GP应用的结果。

GP算法框图如图5.4所示。

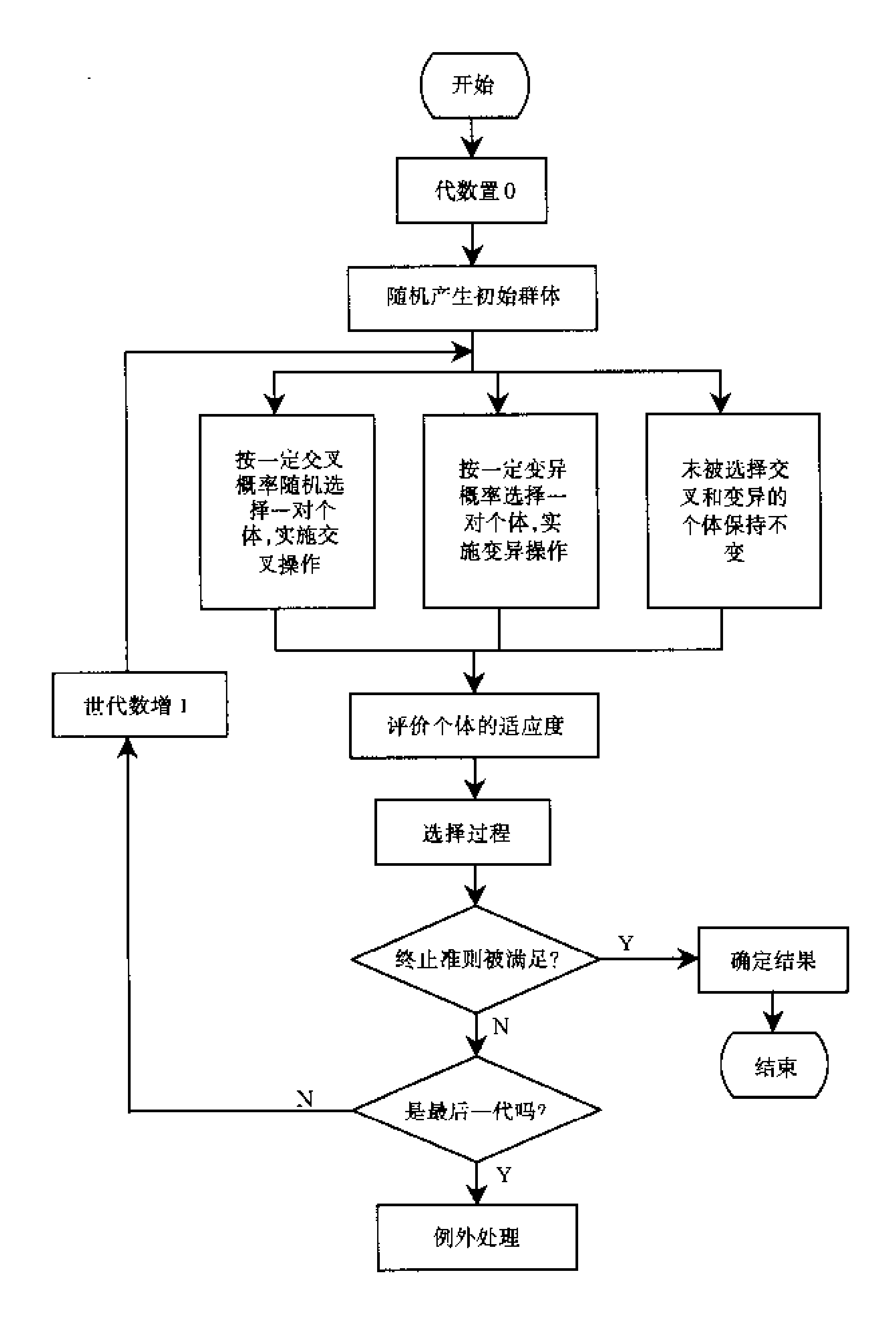


图5.4 GP算法框图

3. 遗传程序设计的遗传操作

GP遗传操作除了包括基本的选择、交叉和变异外,还可以引入一些次级操作来更多地反映遗传程序结构的变化。由于选择操作基本与标准遗传算法类似,这里不再重复。下面重点介绍一下选交叉、变异以及一些次级操作算子的设计问题。

(1) 交叉操作

交叉以如下的方式创造出两个新个体。从当前群体中，根据适应度值挑选两个个体，即父个体，两个父个体的不同部件(如子树、子程序、子路径、子公式等)重新组合产生两个子个体。例如，考虑下面计算机程序(用 LISP 语言表示)：

$$(+\ (*\ 0.234\ z)(-\ x\ 0.789)) \tag{5.11}$$

它表示 $0.234z+x-0.789$，同样考虑第二个程序：

$$(*\ (*\ z\ y)(+\ y\ (*\ 0.314\ z))) \tag{5.12}$$

相当于 $zy(y+0.314z)$。

如图 5.5 所示，将这两个父个体表示为带标号的树，内部结点对应于(+, -, *, /)，终结点(叶结点)对应数据。

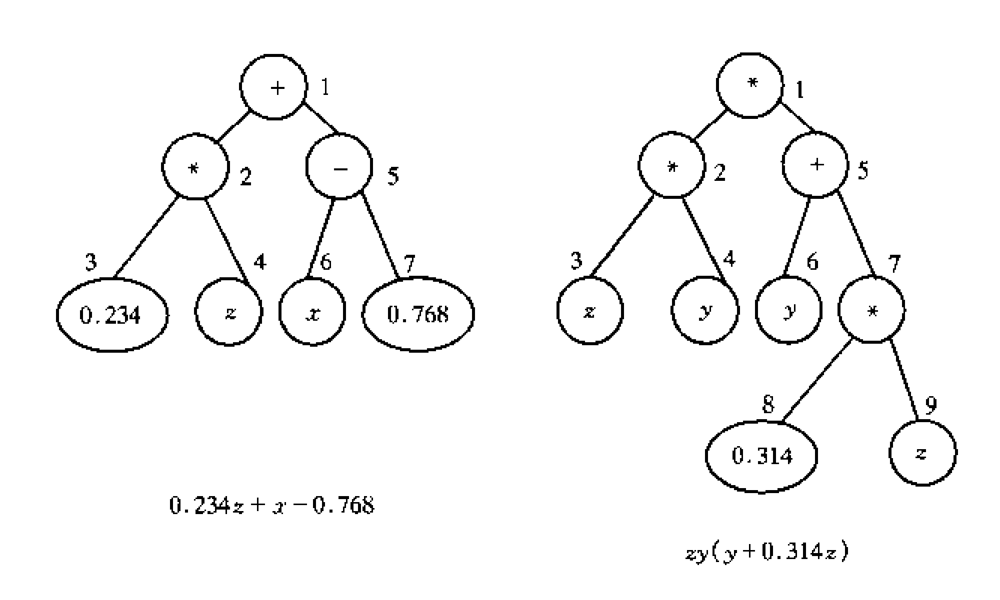


图 5.5 交叉操作的两个父个体

交叉操作通过交换两棵树的子树产生后代，被交换的两棵子树是被随机选择的。例如第一棵树的结点 2 被随机地选择作为第一个父个体的交叉点，第二棵树的结点 5 被随机选择作为第二个父个体的交叉点，两个被交换部分是以交叉点为根的子树。剩余体为父个体去掉交换部分后的剩余部分(remainder)。第一个新个体是把第二个父个体的交换部分插入在第一个父个体的剩余体的交叉点上得到的树；第二个新个体是把第一个父个体的交换部分插入在第二个父个体的剩余体的交叉点上得到的树。两个新个体分别是：

$$(+\ (+\ y\ (*\ 0.314\ z))(-\ x\ 0.768)) \quad 和 \quad (*\ (*\ z\ y)(*\ 0.234\ z)) \tag{5.13}$$

如图 5.6 所示。

交叉操作满足终点集封闭性原则，总是生成合理的下一代的个体。由于父个体是基于适应度挑选出来的，它在解决给定问题时有一定的效果，它们的部件的重新组合产生新一代个体，可能更适合于所给问题。

(2) 变异操作

变异操作用在 GP 中的基本做法是：由程序随机产生一棵新的子树，以代替被突变概率选中结点以下的原有子树部分。如图 5.7 所示。

除了基本变异操作外，还可以采用一种收缩变异操作，其方法是随机选择一个中间结点，将该结点以下的子树部分移到其上一个结点位置，同时删除上一结点的其他子树。如图 5.8

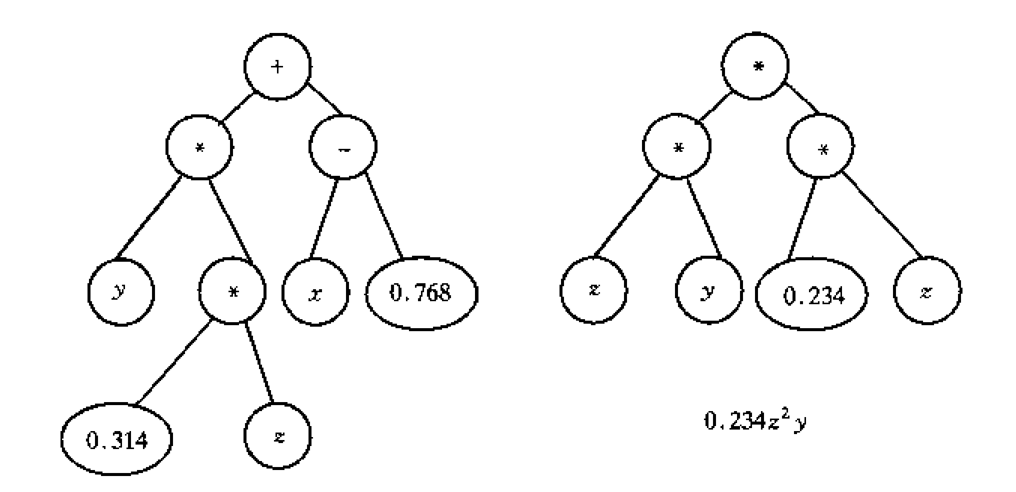


图 5.6 交叉操作产生的两个子个体

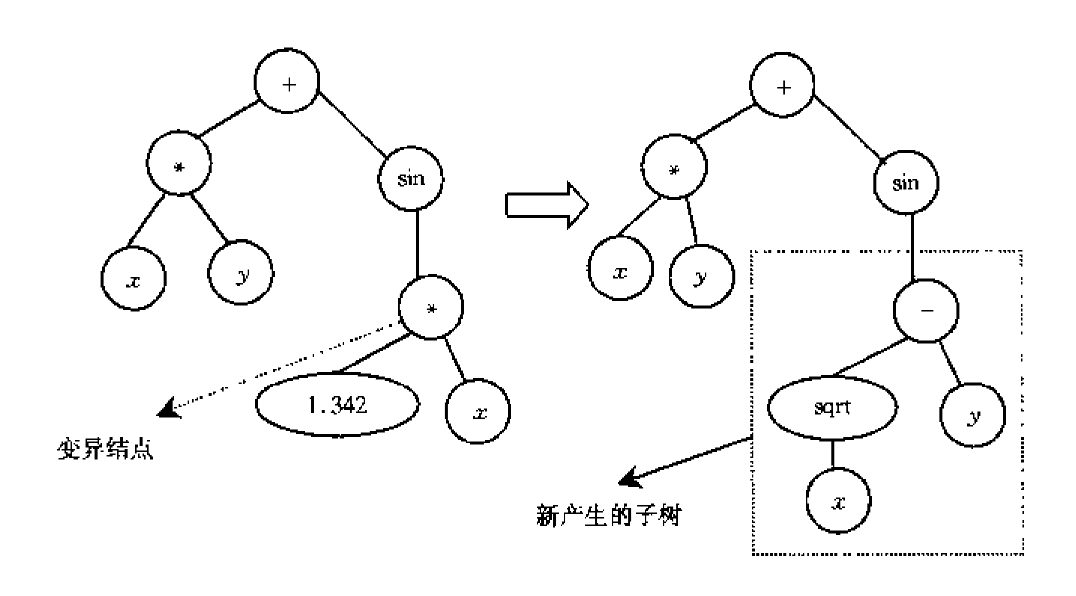


图 5.7 基本变异操作

所示。

(3) **次级操作**

GP 中经常应用两种次级操作,一种是化简操作,它将复杂的结构对应的 S 表达式作适当的简化,例如,

$$(((3+x)*(2-1))+\exp(x-(3*2))) \tag{5.14}$$

可简化为

$$((3+x)+\exp(x-6)) \tag{5.15}$$

另一种是封装(encapsulation)操作,它将进化中出现频次较多的子树封装为用户定义函数,并给它冠名,加入到函数集中备用。它可以比喻一种无性繁殖方式,用于培养好的基因积木块。其示例如图 5.9 所示。图中 $FS(x,y)=\sin(y-\mathrm{sqrt}(x))$。

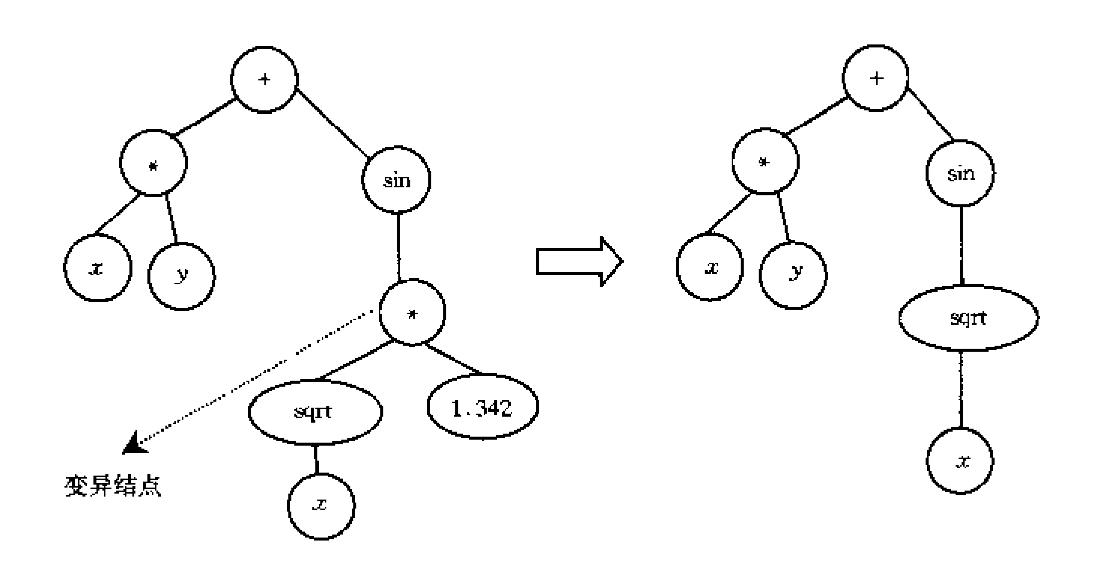


图 5.8　收缩变异操作

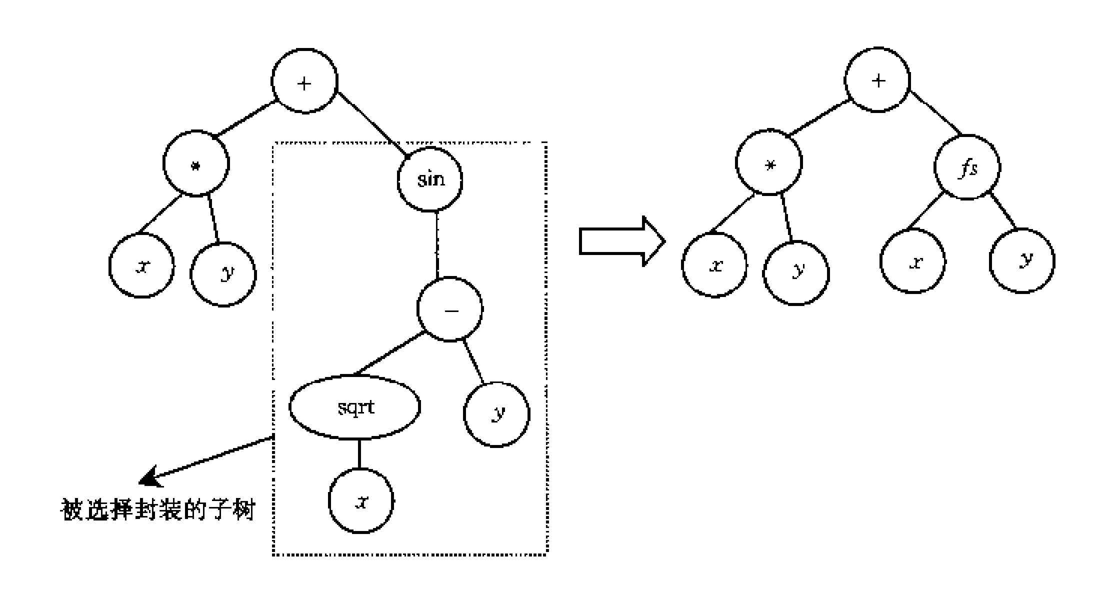


图 5.9　封装操作

4. 适应度评价

GP 中的适应度评价有以下几种形式：

(1) ***原始适应度***

原始适应度通常通过直接计算遗传程序的输出与预定输出之间的绝对误差获得，如下式所示：

$$r(i)=\sum_{j=1}^{N}|S(i,j)-C(j)| \tag{5.16}$$

式中 $S(i,j)$ 为遗传程序 i 在第 j 个输入值时的计算输出，$C(j)$ 为第 j 个输入值对应的预定输出。例如，在符号回归的应用中，假定有 N 个输入-输出样本，$r(i)$ 即为进化获得的回归程序 i 在输入所有样本计算得到的绝对误差和。

(2) ***标准化适应度***

标准化适应度与遗传算法中适应度的最大化要求取得一致，采用如下简单形式：

$$S(i) = r_{max} - r(i) \tag{5.17}$$

式中 r_{max}为最大的原始适应度。

(3) **调整适应度**

调整适应度对标准化适应度作如下修改：

$$a(i) = \frac{1}{1 + S(i)} \tag{5.18}$$

调整后的适应度在(0,1)区间内，比标准适应度形式对最佳个体具有更好的鉴别力。

(4) **正规化适应度**

正规化适应度用于基于适应度比例的选择方法，它可根据调整适应度计算如下：

$$n(i) = \frac{a(i)}{\sum_{k=1}^{M} a(k)} \tag{5.19}$$

式中 M 为种群的大小。

正规化适应度可以直接用作个体的选择概率。

(5) **相关系数**

McKay 建议根据遗传程序计算的预测序列与样本序列的相关系数来计算适应度，对数值预测问题的符号回归更加合适。相关系数的形式如下：

$$c(i) = \left| \frac{\left(\sum_{j=1}^{N} \frac{P(i,j)T(j)}{n(i)} \right) - \left(\sum_{j=1}^{N} \frac{P(i,j)}{n(i)} \right) \left(\sum_{j=1}^{N} \frac{T(j)}{n(i)} \right)}{\sigma_P \sigma_T} \right| \tag{5.20}$$

式中 $P(i,j)$为遗传程序 i 在第j 个样本输入时的计算，$T(j)$为实际第 j 个样本的输出，σ_P，σ_T 分别为预测序列和样本序列的标准差。

5. 初始遗传程序的生成

初始遗传程序的生成过程一般从根结点开始，从原始函数集和运算符中随机选择一个作为根结点，然后根据其变量数或操作符数目产生同样数目的子结点，再在每个子结点做下一层的继续展开，直到所有树分支到达终结点，最终形成一个分层树状结构，代表一个初始遗传程序。

Koza 将遗传程序生成方法按其大小和形式分为以下四种：

(1) **完全生成法**

完全生成法由根结点到终结点顺序生成各个分支等长的遗传程序树。树的深度限制有两种：一种是最大生成深度，是产生初始遗传程序的限制条件；另一种是最大交叉和变异深度，是对遗传操作产生个体时的限制条件。它们的具体数目是依具体问题而定的。完全生成法限制所有的树分支的长度均为最大生成深度，这样只有达到最大深度时才在终结点集中做随机选择，其他结点必须在原始函数集和运算符中选择。

(2) **生长法**

生长法也是由根结点到终结点顺序生成遗传程序树，但每个分支长度可以不相等。在决定中间结点类型时，可以随机地决定属于终结点型还是中间结点型。若选择终结点型，则在终结点集中随机选择其一；若选择中间结点型，则在原始函数和运算符中随机选择其一，这样虽

然存在最大生成深度的限制，但没有一定的必要达到最大深度。

(3) **倾斜生成法**

倾斜生成法将最大生成深度作为种群的一种参数处理。例如，取[2, 6]区间数。当产生50个个体的种群时，先产生最大深度为2的10个个体，然后产生最大深度为3的10个个体，依次类推，最后产生最大深度为6的10个个体。这样种群中的个体结构大小呈均匀分布，倾斜生成法在实际生成遗传程序时可一致地选择完全生成法或生长法。

(4) **倾斜对半生成法**

与倾斜生成法不同的是，在生成每个遗传程序时可以随机地选择采用完全生成法或生长法。Koza认为这种方法最佳，可以产生从大小到形式均有很多变化的种群。

5.4.2 遗传程序设计的应用示例

遗传程序设计的测试通常应用于一个符号回归问题，即预先给定样本数据序列，或者按预定的函数取样计算一个样本序列，用GP方法来获得一些好的遗传程序，从而用发现的函数近似这些样本序列。

下面是一个非线性函数的符号回归问题，原函数为：

$$\mathrm{MIN}(2/x, \mathrm{SIN}(x)+1, \mathrm{EXP}(x)) \qquad x \in (0, 14] \tag{5.21}$$

上述函数在(0, 14]取100个样本，设定种群的大小为70，遗传程序最大深度为6，最大交叉和变异深度为17，交叉概率0.7，变异概率0.001，初始遗传程序采用倾斜对半生成法，原始函数和运算符包括：+，−，*，/，SIN，COS，EXP。适应度评价采用相关系数法，次级遗传操作使用了封装操作。进化终止代数为200。从实算结果中选出两个较优的遗传程序对应的函数，分别如下：

$$\begin{aligned}
F_1 = {} & 1.016\,64-(11.223\,152\,160\,166\,3*(1.052\,085\,605\,373\,03- \\
& 0.314\,700\,854\,700\,854*x+(2*x)/\mathrm{EXP}(x)-\mathrm{COS}(1.035\,439\,940\,059\,38/ \\
& \mathrm{EXP}(3.9/x))))/(-3.924\,801\,130\,187\,8+\mathrm{EXP}(0.872\,744\,507\,645\,751- \\
& \mathrm{SIN}(x))-3.9/x-3.428\,996\,223\,414\,82*x-2*\mathrm{COS}(2*x)- \\
& \mathrm{COS}((\mathrm{EXP}(\mathrm{COS}(\mathrm{EXP}(0.872\,744\,507\,645\,751-\mathrm{SIN}(x))))*\mathrm{SIN}(x))/ \\
& (-0.965\,773\,060\,620\,639+0.54*x))-2*\mathrm{COS}((3.87*\mathrm{SIN}(x))/x))
\end{aligned} \tag{5.22}$$

$$\begin{aligned}
F_2 = {} & -1.020\,47-(0.315\,497*((1.315\,068\,493\,150\,68*x+(4.57+3*x)* \\
& (4.57+3*x+\mathrm{SIN}(0.684\,931\,506\,849\,315*x)))/(4.57+3*x- \\
& \mathrm{COS}(4.57+3*x+\mathrm{SIN}(5.18*(-2.37+x))))-5.18*(5.62+2*x+ \\
& \mathrm{SIN}(x))+\mathrm{SIN}(2.81+0.315\,068\,493\,150\,684*x+2*\mathrm{SIN}(x))+ \\
& \mathrm{SIN}(((4.57+3*x)*(2.81+0.315\,068\,493\,150\,684*x+2*\mathrm{SIN}(x)))/ \\
& (4.57+3*\mathrm{SIN}(0.684\,931\,506\,849\,315*x)-\mathrm{SIN}(x)))))/(2.81+ \\
& 2.315\,068\,493\,150\,68*x-\mathrm{COS}(4.57+x-\mathrm{SIN}(5.18*(2.81+ \\
& 0.315\,068\,493\,150\,684*x))))
\end{aligned} \tag{5.23}$$

图5.10为原函数与进化获得的遗传程序所对应的两个函数的图形。更多的结果表明，遗传程序设计方法在符号回归问题上是有效的，而且显示出很强的解题能力。

我们将在9.2节和9.5节进一步介绍该方法在非线性系统辨识、神经网络学习规则发现问题上的应用。

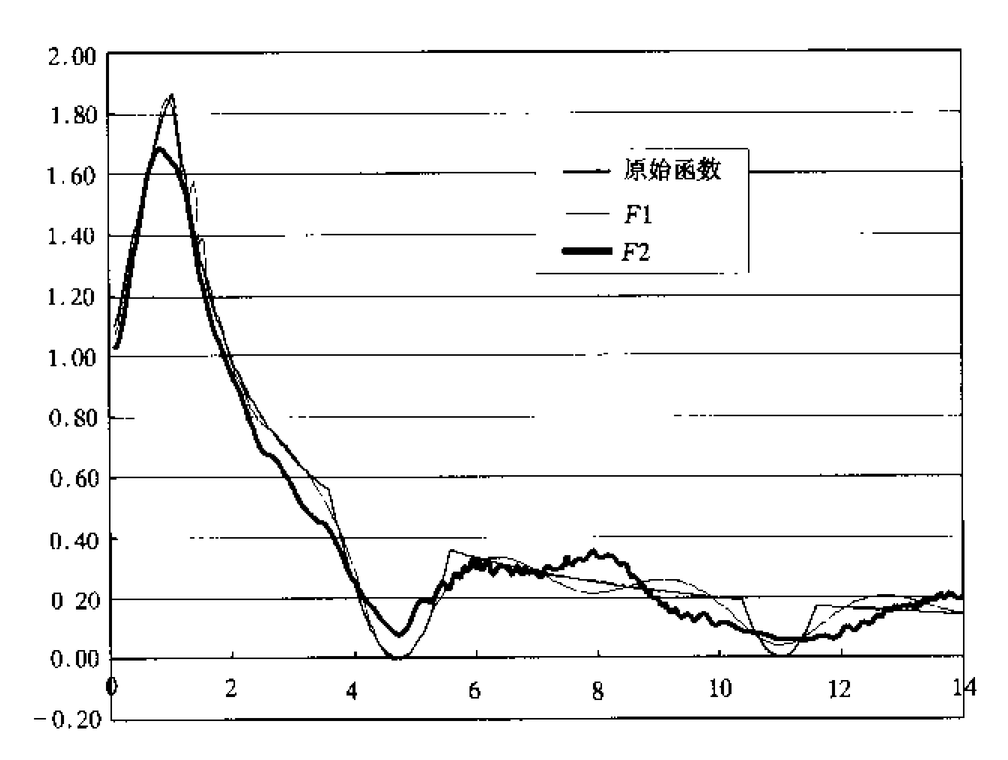


图 5.10　GP 应用于一个符号回归问题

5.4.3　遗传程序设计的主要特点

GP 的特点主要反映在求解、过程和机制诸方面，比较突出的有下列几点：

(1) **自适应性**　自适应性主要是指在计算过程中所获得的可用信息被作为引导搜索路径走向的因素，并据此对算法加以某种调整，以增强其问题求解的能力。

(2) **逼近最优性**　GP 在从搜索开始到当前时刻为止所累计的信息的基础上，以逼近最佳方式对搜索空间中未来轨迹加以定位。

(3) **求解的动态性**　GP 是以动态方式寻找并确定解的结构，这个功能是一个好的机器学习方法所应具备的。解的结构和形态作为求解方法所提供答案的一部分，以非监督学习来获得。而不是被预先给定较强的约束和限制，并且不依赖于问题相关的先验知识。

(4) **求解的表达性**　GP 在可能的计算机程序空间中工作，可以完善的表达方式操作于用户选定的程序搜索空间，并且通常提供了一个与用户表达问题的自然语言形式很接近的相对高层次符号化环境。GP 的个体结构具有符号描述、规则表达和算术表达式等较为灵活的描述能力。

(5) **编程的通用性**　GP 的机理允许通用程序语言的逻辑结构，并且不依赖于符号处理或数值计算形式。

(6) **空间搜索特性**　GP 以随机搜索方式，形成了空间演化轨迹，由于随机机制的引入，使得 GP 的搜索有可能避免 BP 网络中最速下降法调整权值产生的那些缺陷，从整体上进行的结构随机组合，使得结构获得逼近性最优效果。

GP 的思想着眼于程序、结构和问题等智能信息处理中关键性的对象和环节，其程序设计机制有效地吸取了智能逻辑结构、人工智能语言等长处，属于进化计算中很有特色的算法。

5.4.4 遗传程序设计研究的发展趋势

遗传程序设计的研究近年来有了很大的发展,其内容包括理论、算法和应用。1994 年第一届 IEEE 进化计算国际会议就设置了"遗传程序设计"的专题,相应的子标题有理论、分析、控制和扩展。GP 的主要发展趋势可概括为以下几个方面:

(1) **GP 算法研究** 包括问题约束、个体表示、适应度定义、选择策略和遗传算子等具体过程均是重要的内容,与此相关的途径有算法设计、结构描述和数学的形式化建模等,而且常常与具体应用的特点结合起来加以研究,以努力使 GP 有助于实际应用问题的解决。

(2) **GP 的理论研究** 包括借助适应度状态图和模式定理等分析 GP 的自适应机理,利用泛函分析、随机过程和集映射等数学工具建立较为完善的 GP 理论。

(3) **GP 的数据结构研究** J. R. Koza 采用 LISP 在实现 GP 工作中主要的数据结构是"树"。从广义问题求解和程序设计相结合的观点来看,数据结构对于 GP 算法的求解效率具有很大的影响。Perkis 在基于堆栈的 GP 方面的工作就是一个很好的思路,堆栈这种数据结构对于实现递归等过程来说是非常灵活的。

(4) **新概念的构思和相应仿真形式的设计** Handley 适应无环图来表示群体;D' haeseleer 将保存上、下文的交叉操作用于 GP 中;Tackett 和 Carami 构造了孵卵选择的唯一蕴含;伊庭其志、佐藤等将系统辨识技术引入 GP 的算法构造中。ATR 的 H. deGaris 进行人工脑的遗传程序设计研究,该研究旨在元胞自动机内使人工脑以电子速度生成和进化。GMD(德国国家信息处理研究所)的 H. Muhlenbein 等的工作表明增殖遗传程序设计是一种很有希望的方法。他们用这种方法完成了神经网络的综合。

(5) **应用于机器学习、系统动力学分析和随机搜索等方面** 机器学习方面代表性的工作有 Andre 采用 GP 的学习、规划和记忆;Samy Bengio 等学习新规则的寻找;Rosca, Ballard 通过 GP 中的自适应表示;Lay 应用 GP 进行动力学系统多稳态分析;Ulrich Thoneman 则利用 GP 对模拟退火法进行改进。

应该说,从国际学术界的情况看,遗传程序设计的研究在整体水平上还处于比较初步的阶段,但某些方面已获得了可喜的进展,如 GP 在 OCR 系统、虚拟现实、智能信息检索等方面的应用。由于 GP 在软件重新化工程、增加软件的可重用性、CASE 以及进化硬件等方面具有很大的潜力,这为新一代的软件工程和知识工程相结合提供了契机。随着这方面研究的深入,将为人们提供一个崭新的解决复杂设计问题的强有力的工具。

参考文献

[1] Back T, Schwefel H P. Evolution Strategies I: Variants and Their Computational Implementation. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed), Wiley, 1995, 111~126

[2] Schwefel H P, Back T. Evolution Strategies II: Theoretical Aspects. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed), Wiley, 1995, 127~140

[3] Koza J R. Introduction to Genetic Programming, Genetic Programming. Koza J R(ed). 1994, 21~42

[4] O'Reilly T U. An Analysis of Genetic Programming. Doctoral Dissertation, School of Computer Science, Careleton University, Ottawa, Ontario, 1995

[5] Harris C, Buxton B. GP-COM: A Distributed, Component-Based Genetic Programming System in

C++. In: Proceedings of 6th International Conference on Genetic Algorithms, ICGA95, Morgan Kaufmann, 1995, 321～320

[6] Poyhonen H O, Savic D A. Symbolic Regression Using Object－Oriented Genetic Programming(in C++). Research Report, Center for System and Control Engineering, University of Exeter, UK. 1998

[7] Raik S E, Browne D G. Evolving State and Memory in Genetic Programming, In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 73～80

[8] Belding T. The Distributed Genetic Algorithm Revisited. In: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1995, 114～121

[9] Back T, Hoffmeister F, Schwefel H P. A Survey of Evolution Strategies. In: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1991, 2～9

[10] Davidor Y, Schwefel H P. An Introduction to Adaptive Optimization Algorithms Based on Principle of Natural Evolution, Dynamic, and Chaotic Programming. Wiley, 1992

[11] Fogel D B. An Analysis of Evolutionary Programming. In: Proceedings of 1st Annual Conference on Evolutionary Programming, 1992, 43～51

[12] Koza J R. Genetic Programming, MIT Press, 1991

[13] Schwefel H P. Numerical Optimization of Computer Models, Wiley, 1981

[14] Hofbauer J, Sigmund K. The Theory of Evolution and Dynamical Systems. Cambridge University Press, 1988

[15] Collins R J. Studies in Artificial Evolution. Doctoral Dissertation, Artificial Life Laboratory, Department of Computer Science, University of California, 1992

[16] Schoenauer M, Michalewicz Z. Evolutionary Computation at the Edge of Feasibility. In: Parallel Problem Solving from Nature IV, Proceedings of the International Conference of Evolutionary Computation, Berlin, Germany, 1996, 222～231

[17] Yao X. Progress in Evolutionary Computation. In: Lecture Notes in Artificial Intelligence, Springer－Verlag, 1995(956): 155～162

[18] 伊庭斉志. 進化論的計算手法の最近の研究動向. 情報処理, 1998, 39(1): 32～36

[19] 刘健勤, 魏洁敏. 进化计算的可计算性. 计算机技术与自动化, 1998, 17(3): 1～2

[20] 魏洁敏. 一种进化计算的抽象机器模型. 计算机技术与自动化, 1998, 17(3): 3～4

[21] 马丰宁等. 用遗传规划求欧拉回路. 系统工程理论与实践, 1997(5): 19～28

[22] 王正志, 薄涛. 进化计算. 长沙: 国防科技大学出版社, 2000

[23] 刘健勤. 人工生命理论及其应用. 北京: 冶金工业出版社, 1997

[24] 云庆夏. 进化算法. 北京: 冶金工业出版社, 1997

[25] 王宁, 蔚承建, 盛昭翰等. 基于嵌入混沌序列的遗传算法. 系统工程理论与实践, 1999(11): 1～7

[26] 齐越胜, 王保中, 康立山. 遗传程序设计中的赋值函数与节点分类. 计算机科学, 1998, 25(3): 93～94

[27] 陈定君, 周志强, 刘积仁. 遗传程序设计模式理论的新进展. 计算机科学, 1999, 26(7): 14～17

[28] 康立山, 曹宏庆, 陈毓屏. 常微分方程组的演化建模. 计算机学报, 1999, 22(8): 871～876

[29] 叶美盈. 进化策略在控制混沌中的应用. 应用基础与工程科学学报, 1999, 7(2): 139～143

第 6 章 遗传算法与数值优化

最优化问题是遗传算法经典应用领域,但采用常规方法对于大规模、多峰多态函数、含离散变量等问题的有效解决往往存在着许多障碍。遗传算法简单易行、高效性及其普遍适用性,已经赢得了许多应用,并且得到了长足的发展。本章和下一章将重点讨论遗传算法在这一领域的应用。本章首先概述最优化问题的常规方法及其分类;然后介绍 De Jong 的遗传算法应用于含上、下限约束的最优化问题方法以及多峰函数的优化,并引入分享机制;对于含一般线性约束条件的最优化问题,介绍 Michalewicz 的方法;最后考察一下多目标优化问题,引入 Pareto 最优解集的概念,介绍遗传算法在这方面的应用和研究进展。

6.1 最优化问题

一般地,最优化问题(optimization problem)由目标函数(objective function)和约束条件(constraints)两部分构成:

$$\begin{aligned} &\text{Minimize} \quad f(x) = f(x_1, x_2, \cdots, x_n) \\ &\text{subject to} \quad x = (x_1, x_2, \cdots, x_n) \in S \subset X \end{aligned} \tag{6.1}$$

将满足所有约束条件的解空间 S 称为可行域(feasible region),可行域中的解称为可行解(feasible solution);将可行域中使目标函数最小的解称为最优解(optimal solution)。对于最大化问题,可将目标函数乘以(−1),转化为最小化问题求解。

当 $X = \mathbf{R}^n$ 时(n 元实空间),目标函数和约束条件均为线性表达式,最优化问题(6.1)称为线性规划问题(linear programming problem);否则称之为非线性规划问题(nonlinear programming problem)。线性规划问题对应于单纯形法(simplex method)和对偶理论求解。当目标函数 $f(x)$ 为二次函数、约束条件全部为线性表达式时称为二次规划(quadratic programming),可以找到类似于线性规划的在有限步搜索的优化方法。当目标函数 $f(x)$ 为凸函数、可行域为凸空间时,该优化问题称为凸规划(convex programming),依据连续性和可微性的假设,有最小平方和法、最速下降法以及牛顿法等经典无约束方法。对于非凸规划(non−convex programming)问题,虽然可以应用互补转轴理论(complementary pivot theory)的推广,但一直没有非常有效的最优化方法,在一定程度上,随机搜索方法是较好的选择。当 X 或 S 为离散集合构成的解空间时,这类最优化问题称为组合最优化(combinatorial optimization problem)。严格意义上的最优解求取非常困难,研究高速近似的算法是一重要的发展方向。对全局优化问题,目前存在确定性和非确定性两类方法。前者以 Brianin 的下降轨线法、Levy 的隧道法和 R.Ge 的填充函数法为代表。该类方法虽然收敛快、计算效率较高,但算法复杂,求得全局极值的概率不大。非确定性方法以 Monte-Carlo 随机试验法、Hartman 的多始点法、Solis 和 Wets 的结合梯度信息的搜索方法、模拟退火法(simulated annealing)等为代表。该类方法对目标函

数要求低、容易实现、稳定性好,但收敛速度较慢、求得全局极值的概率较低。遗传算法作为现代最优化的手段,实践证明,它应用于大规模、多峰多态函数、含离散变量等情况下的全局优化问题是合适的,在求解速度和质量上远超过常规方法,因而是一高速近似算法。本章和下一章将对遗传算法与常规的一般优化方法进行分析比较。

组合最优化问题,顾名思义,它研究的对象可以看作是在有限集合上定义的函数在各种条件下的极值问题。从理论上说,这类问题若有解,总是可以用枚举的方法找到。这意味着以枚举为基础发展起来的分枝定界法(branch and bound method)具有普遍适用性。但实际上对于大规模的问题分枝定界法通常是不能实际使用的,该方法的计算时间一般为输入数据量(长)的指数函数,计算时间会迅速增加到现代计算工具难以承受的地步。根据计算复杂性的理论,一个好的算法,其计算时间作为输入数据量(长)的函数应该有一个多项式作为上界,这样的算法被称为多项式时间算法,简称有效算法;在组合优化中,至今还没有似乎也不可能发现普遍适用的多项式时间算法。一个多项式时间算法问题被称为多项式时间可解问题,或者P问题。组合优化中多数问题属于一类不知道是否为P问题的问题,其中,包括所谓的NP完全(Nondeterministic Polynomial Completeness)问题。在这类问题里,只要有一个被证明是P问题,那么它们全部是P问题。至今已被证明是属于NP完全问题的数目有几千个之多。一般认为,NP完全问题不会是P问题。因此对NP完全问题,既然没有准确的多项式时间算法,比较现实的妥协方法是采用多项式时间近似算法。近似方法分为启发式方法(heuristic method)和随机方法(random method)两种,启发式方法有有序搜索(贪心算法或称最好优先搜索)和最优 A^* 算法。与盲目搜索不同的是:启发式搜索运用启发信息,应用某些经验或规则来重新排列Open表中结点的顺序,使搜索沿着某个被认为最有希望的前沿区段扩展。启发式方法较多地依赖于对问题构造和性质的认识和经验,适用于解决不太复杂的问题。随机方法不依赖于问题的性质,从解空间 X 中随机地选择多个解,检查这些解的可行性,在可行解集中选择目标函数最小的解作为最优解。这种方法的算法简单明了,但因为需要检查相当多的可行解集,计算非常耗时,尤其对于大范围的搜索,很有可能遗漏最优解。

对于组合最优化问题,与常规方法比较,可以认为遗传算法处于随机方法与启发式方法之间,它在引入随机搜索的同时,在解的构成法和演算过程中考虑了问题的原有构造。

6.2 只含上、下限约束的最优化问题

只含上、下限约束的最优化问题可以描述为:

$$\begin{aligned}&\text{Minimize}\quad f(x_1, x_2, \cdots, x_n)\\&\text{subject to}\quad l_i \leqslant x_i \leqslant u_i \quad i=1,2,\cdots,n\end{aligned}\tag{6.2}$$

遗传算法对于上述优化问题是比较容易的。$x=(x_1, x_2, \cdots, x_n)$的各个变量可以按二进制编码方法分别编码。因此,2.1节介绍的单变量函数在闭区间上的函数优化方法,可以直接推广到多变量的情形。对于变量 x_i 的上、下限约束 $l_i \leqslant x_i \leqslant u_i$ $(i=1,2,\cdots,n)$,依据解的精度要求(有效位数)求得各变量$(x_1, x_2, \cdots, x_n)$的二进制码位数$(m_1, m_2, \cdots, m_n)$,因此将 n 个二进制位串顺序连接起来,构成一个个体的染色体编码,编码的总位数 $m=m_1+m_2+\cdots+m_n$。

图6.1所示的是该优化问题的遗传编码构成。解码时仍按各个变量的编码顺序分别实现常规的二进制编码解码方法即可。除了采用标准的二进制编码方法外,还可以采用Grey编码

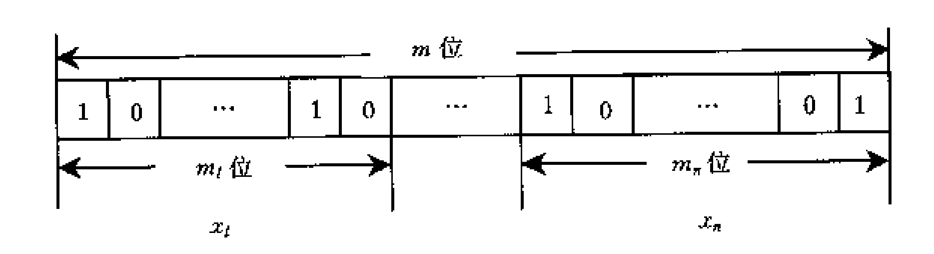


图 6.1 二进制遗传编码示意图

和实数编码方法。1975 年，De Jong 发表了题为"An Analysis of the Behavior of a Class of Genetic Adaptive Systems"的学位论文。主要论述了遗传算法应用于函数最优化的研究。函数最优化问题的复杂性依赖于：局部极小点的数目、局部极小点的分布、局部极小点的函数值的分布以及局部极小点的吸引域等。De Jong 根据函数优化相关的各种特性，如：①连续与非连续，②凸性与非凸性，③单峰性与多峰性，④二次函数与多次函数，⑤低维函数与高维函数，⑥确定的与随机的，从中精心挑选了五种函数最优化的测试例子，如表 6.1 所示。函数 F_1 是简单的平方求和函数，具有一个极小值在点 $x_i=0$；二维 Rosenbrock 函数 F_2 是单峰函数，但它是病态的且极难极小化；F_3 是不连续函数，是由整数阀值的和得到的，在五维空间中有一个极小值；F_4 是有噪声的四次函数；F_5 是多峰函数，具有 25 个局部极小值。图 6.2 是 $F_1 \sim F_5$ 的函数图形示意图。在计算机上经过大量的计算实践，得到了一些在遗传算法的发展和应用过程中具有重要指导意义的结论，其研究方式已成为遗传算法研究和发展的典范。

表 6.1 De Jong 的五种函数最小化问题

函数	上、下限约束
$F_1=\sum_{i=1}^{3} x_i^2$	$-5.12 \leqslant x_i \leqslant 5.12$
$F_2=100(x_1^2-x_2)^2+(1-x_1)^2$	$-2.048 \leqslant x_i \leqslant 2.048$
$F_3=\sum_{i=1}^{5} \text{integer}(x_i)$	$-5.12 \leqslant x_i \leqslant 5.12$
$F_4=\sum_{i=1}^{30} ix_i^4+Gauss(0,1)$	$-1.28 \leqslant x_i \leqslant 1.28$
$F_5=0.002+\sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^2}$	$-65.536 \leqslant x_i \leqslant 65.536$

De Jong 提出的两个评价遗传算法性能的指标：在线性能指标和离线性能指标。

在环境 e 下策略 s 的在线性能(on-line performance)指标 $X_e(s)$ 定义为：

$$X_e(s)=\frac{1}{T}\sum_{t=1}^{T} f_e(t) \tag{6.3}$$

式中，$f_e(t)$ 是在环境 e 下第 t 时刻的平均目标函数值或平均适应度。算法的在线性能指标表示了算法从开始运行一直到当前为止的时间段内性能值的平均值，它反映了算法的动态性能。

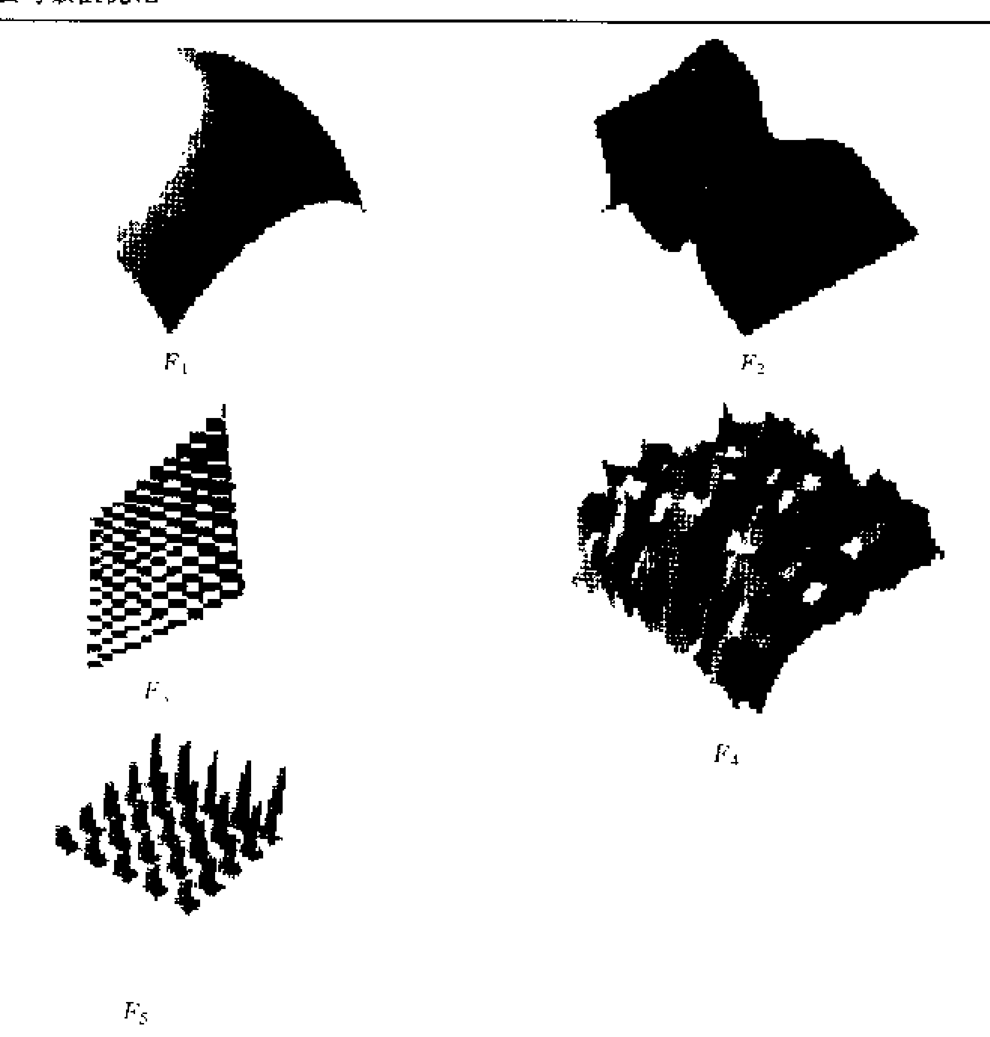


图6.2　$F_1 \sim F_5$ 的函数图形示意图

在环境 e 下策略 s 的离线性能(off-line performance)指标 $X_e^*(s)$ 定义为：

$$X_e^*(s) = \frac{1}{T}\sum_{t=1}^{T} f_e^*(t) \tag{6.4}$$

式中，$f_e^*(t)$ 是在环境 e 下[0, t]时间内最好的目标函数或适应度，$f_e^*(t) = \text{best}\{f_e(1), f_e(1), \cdots, f_e(t)\}$。算法的离线性能表示了算法在运行过程中各进化世代的最佳性能值的累积平均，它反映了算法的收敛性能。

De Jong 采用了以下一些研究方法：

(1) **编码方法**　用二进制编码符号串来表示个体。

(2) **算法的影响参数**　包括了群体大小 M、交叉概率、变异概率、代沟 G，其中代沟 $G(0<G<1)$ 指子代与父代之间的重叠程度。

(3) **算法中的策略 s**　$s1$：基本遗传算法(比例选择、单点交叉、基本位变异)；

$s2$：精英保留模型(elitist model)；

$s3$：期待值模型(expected value model)；

$s4$：精英保留期待值模型(elitist expected value model)；

$s5$：排挤因子模型(crowding model)；

$s6$：广义交叉模型(generalized crossover model)。

经过仔细分析和计算，De Jong 得到了下述几条重要的结论：

结论 1 群体的规模越大，遗传算法的离线性能越好，越容易收敛。

结论 2 规模较大的群体，遗传算法的初始在线性能较差，而规模较小的群体，遗传算法的初始在线性能较好。

结论 3 虽然变异概率的增大也会增加群体的多样性，但它却降低了遗传算法的离线性能和在线性能，并且随着变异概率的增大，遗传算法的性能越来越接近於随机搜索算法的性能。

结论 4 使用精英保留模型和期望值模型的遗传算法比基本遗传算法的性能有明显的改进。

结论 5 对于广义交叉算子，随着交叉点数的增加会降低遗传算法的在线性能和离线性能。

6.3 优化的约束问题解决

6.3.1 约束最优化问题

含不等式约束、等式约束及变量上、下限约束的约束最优化问题标准形式为

$$\text{Minimize } f(x)$$
$$\left.\begin{array}{ll} g_i(x)\leqslant 0, & i=1,\cdots,m \\ h_j(x)\leqslant 0, & j=1,\cdots,l \\ l_i\leqslant x_i\leqslant u_I & \end{array}\right\} \tag{6.6}$$

上述约束最优化问题(Constrained Optimization Problems, COPs)的常规解法可以分为两种途径：一种是把有约束问题化为无约束问题，再用无约束问题的方法去解；另一种是改进无约束问题的方法，使之能用于有约束的情况。第一种途径的历史很悠久，主要是罚函数法(penalty function method)，由 Couran 在 1949 年提出，后来由 Frish(1955)和 Carroll(1959)分别作了发展。罚函数法在实践中使用比较广泛。罚函数法的要点是把问题的约束函数以某种形式归并到目标函数上去，使整个问题变为无约束问题。这种方法对于非线性的约束，设计的算法常常因为迭代点要沿复杂的可行区域边界移动而花费大量的计算而可能导致失败。罚函数法根据解序列相对于原问题的可行性分为外部罚函数法(exterior penalty method)和内部罚函数法(interior penalty method 也称障碍法 barrier function method)。对序列罚函数法性质的进一步研究导出了一次求无约束问题来得到原问题解的恰当罚函数(exact penalty function)。罚函数法的更近发展是乘子法(multiplier method)或 Lagrange 法，以及投影 Lagrange 法。约束最优化问题的第二种途径发展较晚，20 世纪 60 年代 Rosen 对于带线性约束问题提出了著名的梯度投影算法(gradient projection method)。这类算法可以看成是无约束问题中最速下降法在含约束问题上的推广，其基本思想是把负梯度方向投影到可行方向集的一个子集上，取投影为可行下降方向。简约梯度法推广到非线性约束的情况，称为广义简约梯度法(generalized reduced gradient method, GRD)。

将遗传算法应用于约束最优化问题的关键是对约束条件的处理，由于等式约束可以包含到适应度函数中，所以仅需要考虑不等式约束。假设按无约束问题那样求解，在搜索过程中计算由算法产生的目标函数值，并检查是否有约束违反。如果没有违反，则表明是可行解，就根据目标函数值为其指定一个适应值；否则，就是不可行解，因而没有适应度值(适应度值为 0)。

除了许多实际问题是高度约束的之外，这样的处理实际上是行不通的，因为要找到一个可行解同样是很难的，因而需要从不可行解中得到一些信息。借鉴常规方法中的罚函数法是一方便的选择。将罚函数包含到适应度评价中，可以采用下列形式：

$$f(x)+rP(x) \tag{6.7}$$

式中，罚函数 $P(x)$ 为满足下列条件的连续函数：

$$P(x)\begin{cases}=0, & x\in X\\ >0, & x\notin X\end{cases} \tag{6.8}$$

r 为罚函数尺度系数，$r>0$。X 为问题的可行解域。

由于如何设计罚函数以有效地惩罚非可行解，对问题的解决至关重要。罚函数设计方面的研究一直被十分重视，对于不同的问题，人们提出了多种罚函数的形式，如静态罚函数、动态罚函数、退火罚函数、自适应罚函数、启发式罚函数、双重罚函数以及逐次惩罚罚函数等。此外，适于一些高级遗传操作算子也有一定的适用范围，如边界变异(boundary mutation)、启发式交叉(heuristic crossover)、几何交叉(geometrical crossover)、球面交叉(sphere crossover)等。总而言之，罚函数法对于不同的问题需要设计不同的罚函数，而且在约束数目及其复杂性小的情况下才比较适用；对于规模不大的线性约束最优化问题，有较好的应用效果。对于一般的约束处理，通常是很困难的。1997 年，Jan Paredis 提出了共同进化遗传算法(Coevolutionary Genetic Algorithm, CGA)解决一般的约束满足问题，其中一个种群由问题的解组成，另一个种群由约束组成。这两个种群协同进化，较好的解应满足更好的约束，而较优的约束则被更多的解所违背(见 8.4 节)。

下面将介绍 Michalewicz 在遗传算法应用于线性约束最优化问题方面的研究成果。

6.3.2　求解线性约束最优化问题的遗传算法

线性约束最优化问题的一般形式可以描述为：

$$\begin{aligned}&\text{Minimize}\quad f(x_1,\cdots,x_n)\\&\left.\begin{aligned}&\text{subject to}\\&\qquad a_{11}x_1+\cdots+a_{1n}x_n=b_1\\&\qquad\quad\vdots\\&\qquad a_{m1}x_1+\cdots+a_{mn}x_n=b_m\\&\qquad c_{11}x_1+\cdots+c_{1n}x_n\leqslant d_1\\&\qquad\quad\vdots\\&\qquad c_{l1}x_1+\cdots+c_{ln}x_n\leqslant d_l\\&\qquad l_i\leqslant x_i\leqslant u_i,\qquad i=1,\cdots,n\end{aligned}\right\}\end{aligned} \tag{6.9}$$

以上问题的矩阵形式为：

$$\begin{aligned}&\text{Minimize}\quad f(x)\\&\left.\begin{aligned}\text{subject to}\quad &\boldsymbol{Ax}=\boldsymbol{b}\\&\boldsymbol{Cx}\leqslant\boldsymbol{d}\\&\boldsymbol{l}\leqslant\boldsymbol{x}\leqslant\boldsymbol{u}\end{aligned}\right\}\end{aligned} \tag{6.10}$$

式中，$\boldsymbol{x}=(x_1\ \cdots\ x_n)^{\mathrm{T}}$，$\boldsymbol{A}=(a_{ij})$，$\boldsymbol{b}=(b_1\ \cdots\ b_n)^{\mathrm{T}}$，$\boldsymbol{C}=(c_{kj})$，$\boldsymbol{d}=(d_1\ \cdots\ d_n)^{\mathrm{T}}$，$\boldsymbol{l}=(l_1\ \cdots\ l_n)^{\mathrm{T}}$，$\boldsymbol{u}=(u_1\ \cdots\ u_n)^{\mathrm{T}}$，　$(i=1,\cdots,n;\ j=1,\cdots,m;\ k=1,\cdots,l)$。

上述约束优化的目标函数可以是线性函数或者非线性函数。1991 年 Michalewicz 等研究了这类约束最优化问题。通过消除可能的变量以减少变量数目，消除等式约束，并设计特别的遗传操作等手段，使线性约束最优化问题适合于遗传算法求解。这种遗传优化算法称之为 GENOCOP(genetic algorithm for numerical optimization for constrained problem 约束优化的遗传算法)。在介绍这种算法之前，我们给出一个简单的例子说明其基本思想。

下面是一个含 6 个优化变量的线性约束最优化问题：

$$\text{Minimize} \quad f(x_1,x_2,x_3,x_4,x_5,x_6)$$

$$\text{subject to} \quad \left.\begin{array}{l} x_1+x_2+x_3=5 \\ x_4+x_5+x_6=10 \\ x_1+x_4=3 \\ x_2+x_5=4 \\ x_i\geqslant 0 \qquad i=1,\cdots,6 \end{array}\right\} \tag{6.11}$$

首先，根据 4 个独立的等式约束，将变量 x_3,x_4,x_5,x_6 用 x_1,x_2 来表示。

$$\left.\begin{array}{l} x_3=5-x_1-x_2 \\ x_4=3-x_1 \\ x_5=4-x_2 \\ x_6=3+x_1+x_2 \end{array}\right\} \tag{6.12}$$

目标函数表示为：

$$f(x_1,x_2)=f(x_1,x_2,(5-x_1-x_2),(3-x_1),(4-x_2),(3+x_1+x_2)) \tag{6.13}$$

因此，通过消除多余变量和等式约束，约束条件转化为两个变量的不等式约束情形：

$$\left.\begin{array}{l} x_1\geqslant 0,\ x_2\geqslant 0 \\ 5-x_1-x_2\geqslant 0 \\ 3-x_1\geqslant 0 \\ 4-x_2\geqslant 0 \end{array}\right\} \tag{6.14}$$

即

$$\left.\begin{array}{l} 0\leqslant x_1\leqslant 3 \\ 0\leqslant x_2\leqslant 4 \\ x_1+x_2\leqslant 5 \end{array}\right\} \tag{6.15}$$

然后，我们考虑遗传操作的方式。由于采用实数编码方式，变异操作比较简单，检查搜索空间中的一点 $\boldsymbol{x}=(x_1,x_2)=(1.8,2.3)$，保持 x_2 的值使 x_1 发生变化(均匀变异)，变量 x_1 的变化区间为$[0,5-x_2]=[0,2.7]$。在考虑交叉操作时，假设搜索空间中存在两点 $\boldsymbol{x}=(x_1,x_2)=(1.8,2.3)$，$\boldsymbol{x}'=(x_1',x_2')=(0.9,3.5)$，其任意线性组合 $\lambda\boldsymbol{x}+(1-\lambda)\boldsymbol{x}'(0\leqslant\lambda\leqslant 1)$也为搜索空间中的一点，并且满足所有约束条件，从而实现了交叉操作。由于搜索空间是凸性的，设计这样的遗传操作使解向量限制在可行解域内。

一般地，GENOCOP 算法对约束条件处理步骤如下：

(1) **消除等式约束**　假设等式约束 $\boldsymbol{Ax}=\boldsymbol{b}$ 中有 m 个独立的等式，m 个变量 $x_{i_1},x_{i_2},\cdots,x_{i_m}(\{i_1,\cdots,i_m\}\subseteq\{1,2,\cdots,n\}$，可以用剩余的 $n-m$ 个变量表示。等式消除的操作可以描述如下：

将矩阵 $\boldsymbol{A}$ 在第 j 列($j\in\{i_1,\cdots,i_m\}$)处分割为两个分矩阵 $\boldsymbol{A}_1$ 和 $\boldsymbol{A}_2$，类似地分割矩阵 $\boldsymbol{C}$、向量 $\boldsymbol{l}$ 和 $\boldsymbol{u}$，对应分矩阵和向量加下标表示。这样，等式约束成为：

$$\boldsymbol{A}_1\boldsymbol{x}^1+\boldsymbol{A}_2\boldsymbol{x}^2=\boldsymbol{b} \tag{6.16}$$

由于 $\boldsymbol{A}_1^{-1}$存在，$\boldsymbol{x}^1=\boldsymbol{A}_1^{-1}\boldsymbol{b}-\boldsymbol{A}_1^{-1}\boldsymbol{A}_2\boldsymbol{x}^2$，这样，变量 $x_{i_1},x_{i_2},\cdots,x_{i_m}$ 可用剩余的变量的线性组合表示。对于变量 $x_{i_j}(j=1,\cdots,m)$的上、下限约束 $l_{i_j}\leqslant x_{i_j}\leqslant u_{i_j}$，去掉其中所有的 x_{i_j}，有下列新的不等式成立：

$$\boldsymbol{l}_1\leqslant\boldsymbol{A}_1^{-1}\boldsymbol{b}-\boldsymbol{A}_1^{-1}\boldsymbol{A}_2\boldsymbol{x}^2\leqslant\boldsymbol{u}_1 \tag{6.17}$$

加上原问题中不等式 $\boldsymbol{Cx}\leqslant\boldsymbol{d}$，即：

$$\boldsymbol{C}_1\boldsymbol{x}^1+\boldsymbol{C}_2\boldsymbol{x}^2\leqslant\boldsymbol{d} \tag{6.18}$$

将 $\boldsymbol{x}^1$ 代入上式转化为：

$$\boldsymbol{C}_1(\boldsymbol{A}_1^{-1}\boldsymbol{b}-\boldsymbol{A}_1^{-1}\boldsymbol{A}_2\boldsymbol{x}^2)+\boldsymbol{C}_2\boldsymbol{x}^2\leqslant\boldsymbol{d} \tag{6.19}$$

因此，将 m 个变量 $x_{i_1},x_{i_2},\cdots,x_{i_m}$ 消除后，最终的约束由以下的不等式约束组成：

① 原上下限约束　$\boldsymbol{l}_2\leqslant\boldsymbol{x}^2\leqslant\boldsymbol{u}_2$　(6.20)

② 新增加的不等式约束　$\boldsymbol{A}_1\boldsymbol{l}_1\leqslant\boldsymbol{b}-\boldsymbol{A}_2\boldsymbol{x}^2\leqslant\boldsymbol{A}_1\boldsymbol{u}_1$　(6.21)

③ 原不等式　$(\boldsymbol{C}_2-\boldsymbol{C}_1\boldsymbol{A}_1^{-1}\boldsymbol{A}_2)\boldsymbol{x}^2\leqslant\boldsymbol{d}-\boldsymbol{C}_1\boldsymbol{A}_1^{-1}\boldsymbol{b}$　(6.22)

(2) **GENOCOP 算法**　为将 GENOCOP 算法与常规约束最优化方法比较，Michalewicz 等考虑了下面的 7×7 的运输规划问题作为实例。

$$
\begin{aligned}
&\text{Minimize}\quad f(\boldsymbol{x})=f(x_1,x_2,\cdots,x_{49})\\
&\text{subject to}\\
&\left.\begin{aligned}
&x_1+x_2+x_3+x_4+x_5+x_6+x_7=27\\
&x_8+x_9+x_{10}+x_{11}+x_{12}+x_{13}+x_{14}=28\\
&x_{15}+x_{16}+x_{17}+x_{18}+x_{19}+x_{20}+x_{21}=25\\
&x_{22}+x_{23}+x_{24}+x_{25}+x_{26}+x_{27}+x_{28}=20\\
&x_{29}+x_{30}+x_{31}+x_{32}+x_{33}+x_{34}+x_{35}=20\\
&x_{36}+x_{37}+x_{38}+x_{39}+x_{40}+x_{41}+x_{42}=20\\
&x_{43}+x_{44}+x_{45}+x_{46}+x_{47}+x_{48}+x_{49}=20\\
&x_1+x_8+x_{15}+x_{22}+x_{29}+x_{36}+x_{43}=20\\
&x_2+x_9+x_{16}+x_{23}+x_{30}+x_{37}+x_{44}=20\\
&x_3+x_{10}+x_{17}+x_{24}+x_{31}+x_{38}+x_{45}=20\\
&x_4+x_{11}+x_{18}+x_{25}+x_{32}+x_{39}+x_{46}=20\\
&x_5+x_{12}+x_{19}+x_{26}+x_{33}+x_{40}+x_{47}=20\\
&x_6+x_{13}+x_{20}+x_{27}+x_{34}+x_{41}+x_{48}=20\\
&x_7+x_{14}+x_{21}+x_{28}+x_{35}+x_{42}+x_{49}=20\\
&\qquad x_i\geqslant 0,\quad i=1,2,\cdots,49
\end{aligned}\right\}
\end{aligned}
\tag{6.23}
$$

对于非线性目标函数的构造，考虑下面几种测试函数 $A(x)$～$F(x)$(如图 6.3 所示)。

① 函数 $A(x)$

$$A(x)=\begin{cases}0 & 0<x\leqslant S\\ c_{ij} & S<x\leqslant 2S\\ 2c_{ij} & 2S<x\leqslant 3S\\ 3c_{ij} & 3S<x\leqslant 4S\\ 4c_{ij} & 4S<x\leqslant 5S\\ 5c_{ij} & 5S<x\end{cases} \tag{6.24}$$

式中，S 小于一个典型的 x 值。

② 函数 $B(x)$

$$B(x)=\begin{cases}c_{ij}\dfrac{x}{S} & 0<x\leqslant S\\ c_{ij} & S<x\leqslant 2S\\ c_{ij}(1+\dfrac{x-2S}{S}) & 2S<x\end{cases} \tag{6.25}$$

式中，S 和典型的 x 具有同样的阶。

③ 函数 $C(x)$

$$C(x)=c_{ij}x^2 \tag{6.26}$$

④ 函数 $D(x)$

$$D(x)=c_{ij}\sqrt{x} \tag{6.27}$$

⑤ 函数 $E(x)$

$$E(x)=c_{ij}\left(\frac{1}{1+(x-2S)^2}+\frac{1}{1+(x-\frac{9}{4}S)^2}+\frac{1}{1+(x-\frac{7}{4}S)^2}\right) \tag{6.28}$$

式中，S 和典型的 x 具有同样的阶。

⑥ 函数 $F(x)$

$$F(x)=c_{ij}x(\sin(x\frac{5\pi}{4S})+1) \tag{6.29}$$

式中，S 和典型的 x 具有同样的阶。

非线性运输问题的目标函数为：

$$\sum_{i,j}f(x_{ij})+P \tag{6.30}$$

这里，$f(x_{ij})$取测试函数 $A(x)\sim F(x)$中任一函数，P 为罚函数。

$$P=k\cdot(\frac{t}{T})^p\cdot\bar{f}\cdot\sum_{i=1}^{14}d_i \tag{6.31}$$

式中，$\bar{f}$ 为第 t 代群体的平均适应度。k 和 p 为参数，取 $k=1$，$p=1/14$。T 为最大运行代数，d_i 为第 i 个约束的违反度。

对于约束 $\sum_{i\in W}x_i=val$，$W\subseteq\{1,\cdots,49\}$，个体的染色体表示为$(v_1,\cdots,v_{49})$，其约束违反度定义为：

$$d_i=\left|\sum_{i\in W}v_i-val\right| \tag{6.32}$$

7×7 费用参数 c_{ij}取表 6.2 所列数据。

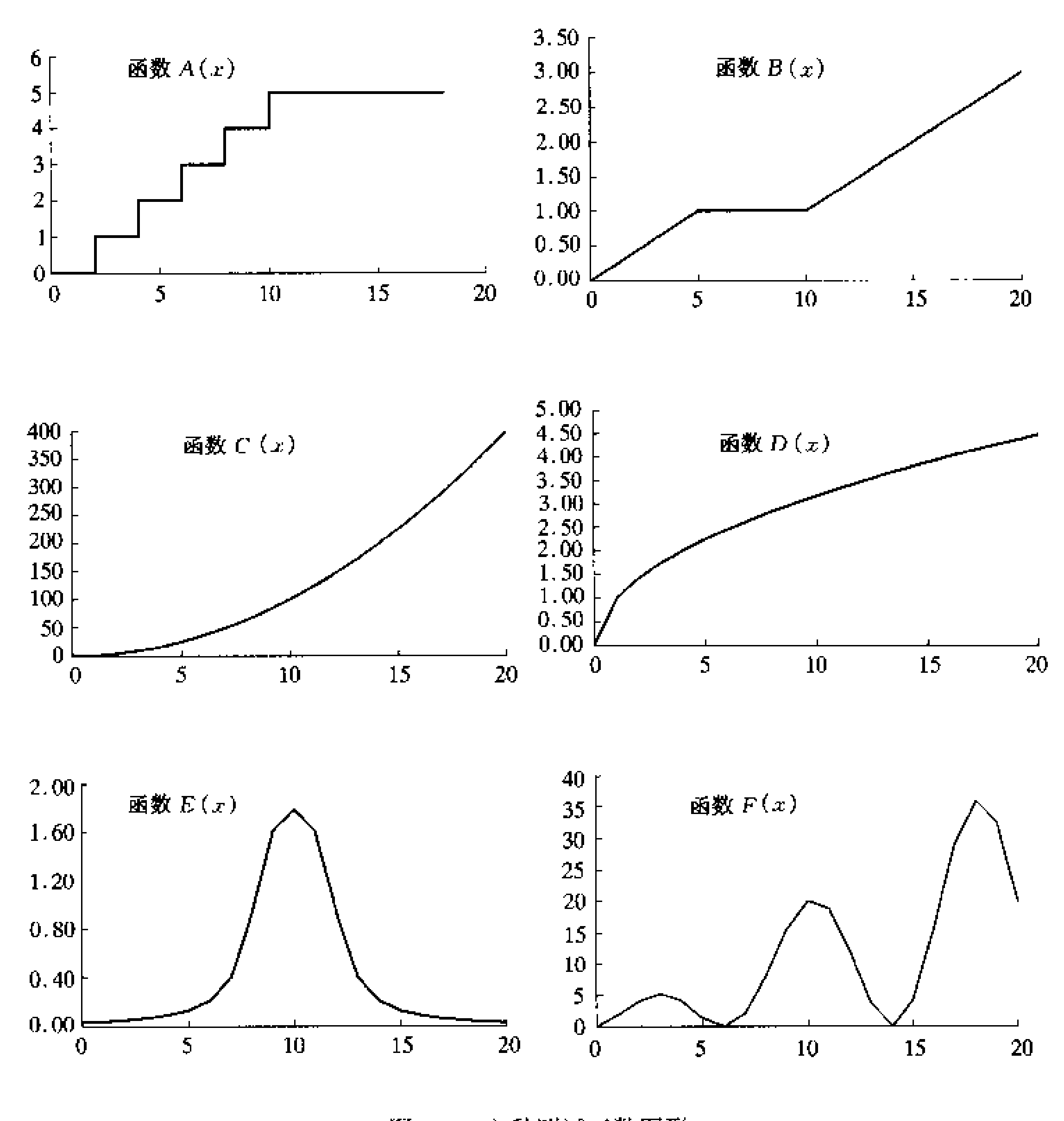


图 6.3　六种测试函数图形

表 6.2　7×7 费用参数表

	27	28	25	20	20	20	20
20	0	21	50	62	93	77	1 000
20	21	0	17	54	67	1 000	48
20	50	17	0	60	98	67	25
23	62	54	60	0	27	1 000	38
26	93	67	98	27	0	47	42
25	77	1 000	67	1 000	47	0	35
26	1 000	48	25	38	42	35	0

为推导 S，需估计 x 的典型值。可通过初步运行的方式估算 x_{ij}的数目和大小。用这种方式可估算每个弧线的平均流量，找到 S 的值。对于函数 A 使用 $S=2$，对于 B，E 和 F 使用 $S=5$。

由于该规划问题含有 13 个独立的等式约束,因此,可以消除 13 个变量。被消除的 x_1, $x_2,\cdots,x_8,x_{15},x_{22},x_{36},x_{44}$,其余的 36 个变量按顺序设定为 $y_1,y_2,\cdots,y_{36}$, $y_1=x_9$, $y_2=x_{10}$, $\cdots$, $y_{36}=x_{49}$。按照等式约束消除的方法,将原规划问题转化为:

$$0 \leqslant y_1 \leqslant 20 \tag{6.33}$$

$$93-\sum_{i=2}^{30} y_i + y_{31} \leqslant y_1 \leqslant 113 + y_{31} - \sum_{i=2}^{30} y_i \tag{6.34}$$

$$\sum_{i=31}^{36} y_i - 20 - y_7 - y_{13} - y_{19} - y_{25} \leqslant y_1 \leqslant \sum_{i=31}^{36} y_i - 20 - y_7 - y_{13} - y_{19} - y_{25} \tag{6.35}$$

分别以 $A(x)\sim F(x)$测试函数构成的目标函数,设定群体大小 40,均匀变异概率 0.08,边界变异概率 0.03,非均匀变异概率 0.07,简单交叉概率 0.10,单一算术交叉概率 0.10,全体交叉概率 0.10,实现 GENOCOP 算法,进行演算 8 000 代的试验。表 6.3 所示列出了中间世代优化结果。

表 6.3 中间世代优化结果

函数	世代					
	1	500	1 000	2 000	4 000	8 000
A	1 085.8	273.4	230.5	153.0	94.5	24.15
B	932.4	410.6	258.4	250.3	209.2	205.60
C	83 079.1	14 122.7	4 139.0	2 944.1	2 772.7	2 571.04
D	1 733.6	575.3	575.3	480.2	480.2	480.16
E	225.2	204.9	204.9	204.9	204.9	204.82
F	3 799.3	1 719.0	550.8	320.9	166.4	119.61

对于非线性目标函数的运输问题,获取大范围全局最优解必须满足目标函数为凸性的要求,而采用常规方法容易收敛于局部最优解。Michalewicz 等将基于拟牛顿法的非线性最优化算法 GAMS 与 GENOCOP 算法进行比较,比较结果见表 6.4。

表 6.4 GAMS 与 GENOCOP 算法比较

函数	GAMS	GENOCOP	误差%
A	96.00	24.15	297.52
B	1 141.60	205.60	455.25
C	2 535.29	2 571.04	-1.41
D	565.15	480.16	17.70
E	208.25	204.82	1.67
F	43 527.54	119.61	36 291.22

从计算结果分析,GENOCOP 算法比商品化的软件包 GAMS 获得的结果要理想。有趣的是,对于费用函数 $A(x)$, $B(x)$以及特别“不规则的”费用函数 $F(x)$, GENOCOP 算法要好得多。而对于其他费用函数 $C(x)$, $D(x)$和 $E(x)$,两种方法的结果比较相近。

6.4 多目标优化问题

工程中经常会遇到在多准则或多设计目标下设计和决策的问题,如果这些目标是相背的,需要找到满足这些目标的最佳设计方案。解决含多目标和多约束的优化问题,即多目标优化(Multi-Objective Optimization, MO)。通常的做法是根据某效用函数将多目标合成单一目标来进行优化。但大多数情况下,在优化之前这种效用函数是难以确知的。这样为了使决策者深入掌握优化问题的特点,有必要提供多个解以便于作出合理的最终选择。

法国经济学家 V. Pareto(1848~1923)最早研究经济领域内的多目标优化问题,他的理论被称为 Pareto 最优性理论。用求单目标优化的方法求最优解,获得的所谓理想解往往在可行域之外。MO 问题需要优化一组费用函数,其解不是单一点,而是一组点的集合,称之为 Pareto 最优集(Pareto - optimal set)。Pareto 最优集定义如下:

对于最小化 MO 问题, n 个目标分量 $f_k(k=1,\cdots,n)$ 组成的向量 $\bar{f}(\bar{x})=(f_1(\bar{x}),f_2(\bar{x}),\cdots,f_n(\bar{x}))$ 其中 $x_u\in U$ 为决策变量,若 $\bar{x}_u$ 为 Pareto 最优解满足:

当且仅当,不存在决策变量 $\bar{x}_v\in U$, $v=f(\bar{x}_v)=(v_1,\cdots,v_n)$ 支配 $u=f(\bar{x}_u)=(u_1,\cdots,u_n)$ 即不存在 $\bar{x}_v\in U$ 使得下式成立:

$$\forall i\in\{1,\cdots,n\},v_i\leqslant u_i\ \wedge\ \exists i\in\{1,\cdots,n\}\mid v_i<u_i \tag{6.36}$$

常规 MO 问题求解方法有多目标加权法、层次优化法、ε-约束法、全局准则法、目标规划法等,其中以目标规划法(goal programming)最为著名。这些算法的特点是将多目标转化为单目标处理,往往只能得到一个解。除了预先获知目标函数最优值的情况外,不能保证 Pareto 最优性,即使最优化求解很成功。

图 6.4 所示的是 MO 问题解集空间示意图, Pareto 最优集的每个解都是 MO 问题的一个非劣解。遗传算法通过代表整个解集的种群进化,以内在并行的方式搜索多个非劣解,决策者可以在多个解中选择决策方案,这对于解决 MO 问题是非常诱人的。下面在分析适用于多目标优化的遗传算法基础上,讨论一种结合决策偏好与 Pareto 秩的多目标遗传算法(Multi -

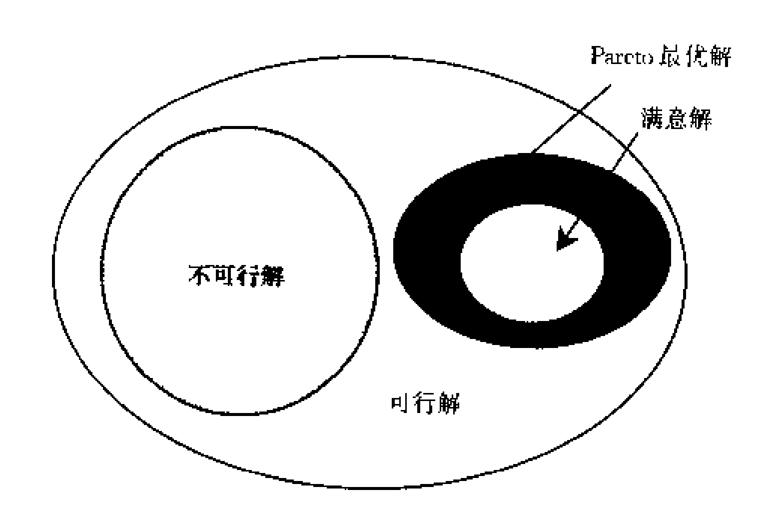


图 6.4 MO 问题解集空间示意图

Objective Genetic Algorithms, MOGA)。

6.4.1 多目标优化的遗传算法

1. 非 Pareto 方法

Schaffer(1985)研究多目标优化时，在扩展 SGA(Simple Genetic Algorithms)时提出了向量形式的适应度计算方法，称为向量评价的遗传算法(Vector Evaluated Genetic Algorithm, VEGA)。其选择方法作了修改，在每一代，基于各目标函数的计算，用适应度比例法产生一定数目的子种群。假定种群的大小为 N，目标函数个数为 q，将产生 q 个子种群，子种群大小为 N/q。然后将其混合起来形成新的一代，继续执行交叉和变异的遗传操作，VEGA 方法如图 6.5 所示。Richardson 等指出这种将所有个体混合起来的做法等价于将适应度函数线性求和，只不过权重取决于当前的世代。

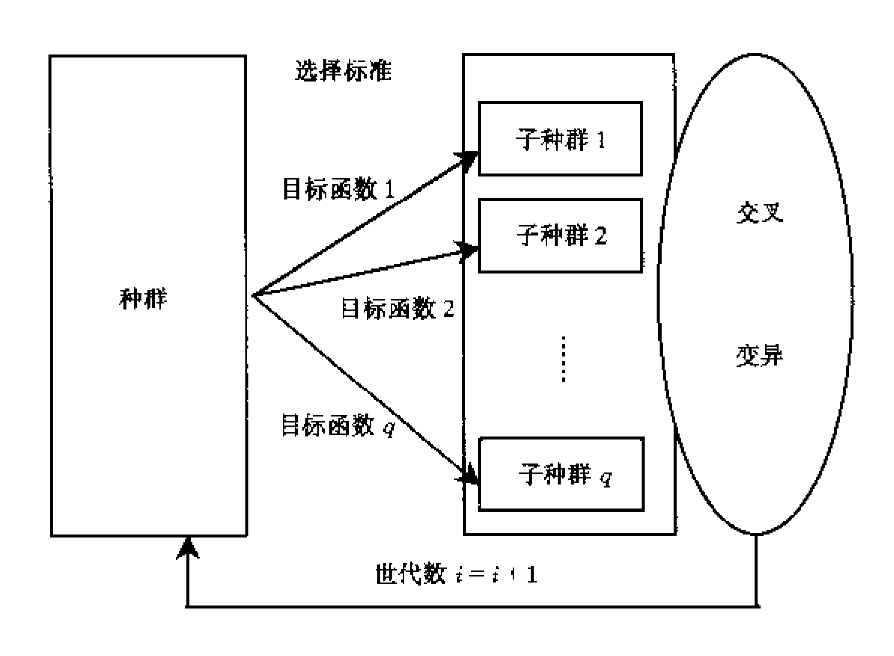


图 6.5 VEGA 方法

2. Pareto 方法

Goldberg [3]提出来一种 Pareto 方法，其基本想法是将多个目标值直接映射到适应度函数中，于是在 Pareto 最优集的基础上提出了一种基于秩的适应度函数形式，先将多目标函数值组成一个向量代表一个个体，假定个体 x_i 在 t 代时种群中有 $p_i^{(t)}$ 个个体支配于它，则它在种群内的秩为：$rank(x_i, t) = 1 + p_i^t$。如图 6.6 所示，所有无支配个体的秩为 1，个体 3 劣于个体 2，因为个体 2 落在折衷区域内部。

关于适应度计算，注意到在某一世代并非所有的秩都有必要存在，图中序位 4 是缺失的。因此，按秩对适应度计算方法作如下调整：

① 种群内个体进行排序；

② 按线性插值的方法计算个体适应度，序位为 1 者为最优个体。

计算具有相同秩的个体平均适应度值，以便它们具有相同的选中概率。这样在适当选择压力的前提下可以保持整个种群的适应度值为常量。这种方法的关键是求各个体的秩，在秩的基础上得到的适应度是均匀分布的，但求秩的过程一般十分复杂，运算量大，特别当种群规

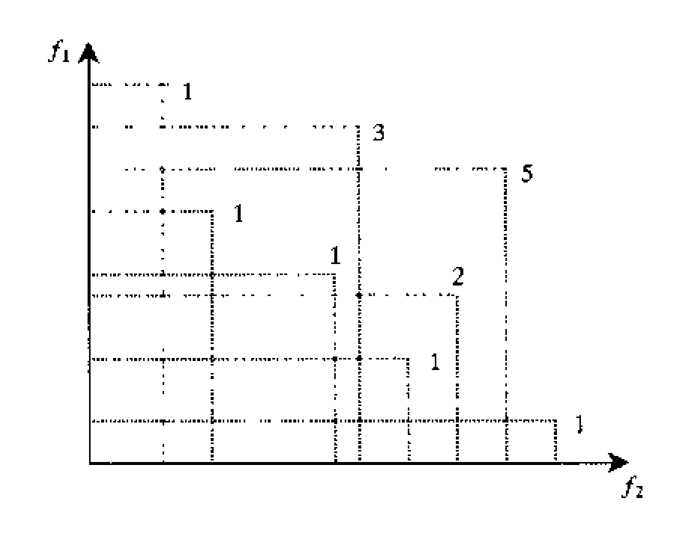


图6.6 种群中个体的Pareto秩

模很大时计算耗时多。

J. Horn和N. Nafpliotis提出了不同的选择方法，有点类似于竞争选择法。在种群中随机地挑选两个候选个体，在种群中另外随机地挑选一个个体参照集。然后候选个体一一与参照集中的个体作比较。如果一个个体劣于参照集，另一个优于参照集，则后者被选作为再生对象。如果两个个体均优于或均劣于参照集，必须采用分享法(sharing method)以确定哪一个是选中的个体。但并非所有情况下都要在给定的候选集中决定出一个个体。例如两个个体正好处于当前的无支配前沿(Non-dominated frontier)时两个个体之间不存在优劣。为了防止种群收敛到Pareto前沿的单一区域，当两个个体之间无偏好时引入一种适应度分享法，保持Pareto前沿的遗传多样性。分享操作目的是减少相似个体的复制量以尽可能保持种群的多样性，从而达到同时探索多个区域的目的。该算法被称为Niched Pareto Genetic Algorithms，即NPGA。

3．结合目标值及其优先级偏好信息的Pareto方法

Charnes和Cooper(1961)以及后来的Ijiri(1965)对目标规划法进行了系统研究。决策者预先为各目标函数确定好目标值，目标值可以是理想值也可以不是理想值，然后按照目标的重要程度给一个权重系数。优化的目标函数被表示成目标偏差值的最小化。目标约束区别于一般的约束条件，它实际上是决策者预期满足的，但微小的偏差也是可以接受的。受目标规划的影响，在MO问题中目标和目标优先级是比较容易得到的偏好信息。不同的目标给定相同的优先级，可以避免使用目标函数的距离测度，而距离测度不可避免地依赖于具体问题中给定目标值的大小。Carlos M. Fonseca和Peter J. Fleming提出了结合偏好信息的关系算子，整个种群的个体排序按照定义的关系算于进行。

与单目标的情况相反，多目标排序不是唯一的。这是由于支配性和偏好性的概念只定义了全局秩没有定义部分秩。两个目标 f_1 和 f_2 的目标偏好值分别设置为 g_1 和 g_2，当两个目标优先级相同时，个体的排序结果如图6.7(a)；当 f_2 比 f_1 的优先级高时，个体的排序结果如图6.7(b)所示。这种方法依赖于决策者提供的目标值及其优先级偏好信息，在某种程度上仍然取决于决策者对问题的把握程度，最终得到的一组Pareto最优解在实际问题中不会是等价的，还需要决策者来遴选。

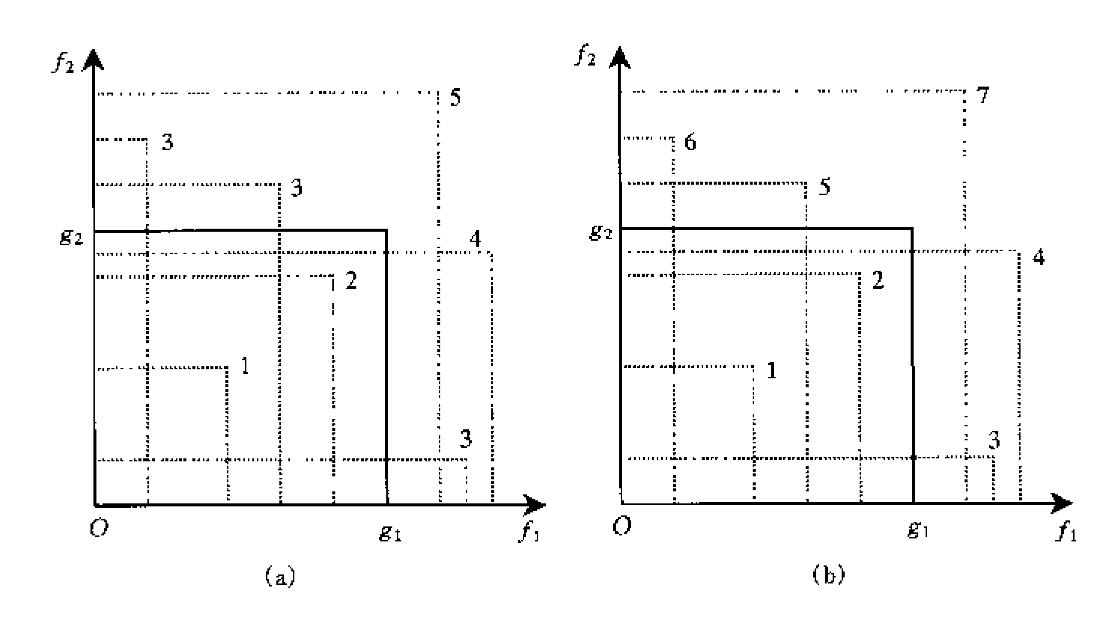


图 6.7 种群中个体的 Pareto 秩

(a) f_1 与 f_2 优先级相同； (b) f_2 比 f_1 优先级大

6.4.2 结合非精确目标权重偏好信息与 Pareto 秩的交互式多目标遗传算法

我们知道,多目标加权法将每个目标函数值乘上一个权重,然后加起来作为一个目标,采用单目标方法优化求取最优解。目标的权重作为决策者的一种偏好信息,一般很难预先确定。因此,我们提出由决策者交互优化过程产生非精确的目标权重引导遗传搜索的一种新的多目标遗传算法。根据决策者预先比较为数较少的可行解,获得非精确的权重信息,然后按照一般的 Pareto 秩构造适应度函数,进行遗传操作,逐代搜索获得无支配前沿,最终获得满足决策偏好的满意解,而不是一组无所适从的 Pareto 最优解。

对目标 $a_k(k=1,2,\cdots,q)$正规化处理,设 $a_k^{\max}$ 和 $a_k^{\min}$ 分别为目标a_k 最大值和最小值,若 a_k 为效益型,线性正规化后目标值 $v_k(a_k)$为:

$$v_k(a_k) = \frac{a_k - a_k^{\min}}{a_k^{\max} - a_k^{\min}} \tag{6.37}$$

若 a_k 为成本型,线性正规化后目标值 $v_k(a_k)$为:

$$v_k(a_k) = \frac{a_k - a_k^{\max}}{a_k^{\min} - a_k^{\max}} \tag{6.38}$$

可行解 x 的多目标函数为:

$$v_x = \sum_k w_k v_k(a_k)_x \qquad \text{其中}, w_k > 0, \sum_{k=1}^{q} w_k = 1 \tag{6.39}$$

若已知两个可行解 x_1 和 x_2 且 $x_1 \succ x_2$ 则 $V_{x_1} > V_{x_2}$,即

$$\sum_{k=1}^{q} w_k[v(a_k)_{x_1} - v(a_k)_{x_2}] > 0 \tag{6.40}$$

另一可行解 x_3,欲判断 x_3 与 x_1 之优劣,只需求解下面线性规划问题:

最小化 $$z = \sum_{k=1}^{q} w_k[v(a_k)_{x_3} - v(a_k)_{x_2}] \tag{6.41}$$

约束条件
$$\sum_{k=1}^{q} w_k [v(a_k)_{x_1} - v(a_k)_{x_2}] > 0 \tag{6.42}$$

$$w_k > 0, \quad \sum_{k=1}^{q} w_k = 1 \tag{6.43}$$

若以上规划问题解存在，则有：

$$z_{\min}\begin{cases} > 0 & x_3 \succ x_1 \\ = 0 & x_3 \sim x_1 \\ < 0 & x_3 \prec x_1 \end{cases} \tag{6.44}$$

在上述判别中权重作 $w_k (k = 1, 2, \cdots, q)$ 为一种非精确偏好信息，包含在可行解的比较中，我们称之为非精确偏好包含。据此构造一种交互式多目标遗传算法：

第1步　随机产生代表若干个可行解的初始种群，将个体的目标值作正规化处理。

第2步　有差异地挑选几个个体，由决策者进行比较判别优劣性，产生一组非精确偏好包含的约束。若不能进行有差异的挑选，即认为已经获得一组满意解，即停止。

第3步　建立线性规划模型的进行个体比较，对当前种群的个体进行排序。

第4步　依据个体的 Pareto 秩，计算适应度值，并进行分享操作。

第5步　选择个体，完成交叉、变异的遗传操作，产生新一代个体。

第6步　世代更迭，每隔一定代数，需要执行非精确偏好包含，转第2步。

第7步　若代数超过一定数目，则停止；否则转向第3步。

6.4.3　一个多目标优化问题计算实例

下面以 J. Horn 和 N. Nafpliotis 提出的一个简单多目标优化问题为例，假定 S_l 为固定长度 l 的二进制串，对 S_l 而言有两个目标，一是串中含1的个数，表示为 $U[S_l]$，二是串中10和01的对数，表示为 $P[S_l]$。例如 $S_8 = 11110101$，有 $U[S_8] = 6$，$P[S_8] = 4$。该问题是给定串长度，求取串变量使 $U[S_l]$ 和 $P[S_l]$ 同时最大化。取 Niche 半径为5，选择方法为锦标赛选择，交叉率为0.7，变异率为0.3，按 NPGA 算法，第200代个体分布统计如图6.8所示(种群大小为400)，最终解几乎是 Pareto 最优解的全集。设定串长28，种群大小为100，开始在初始群体中选择

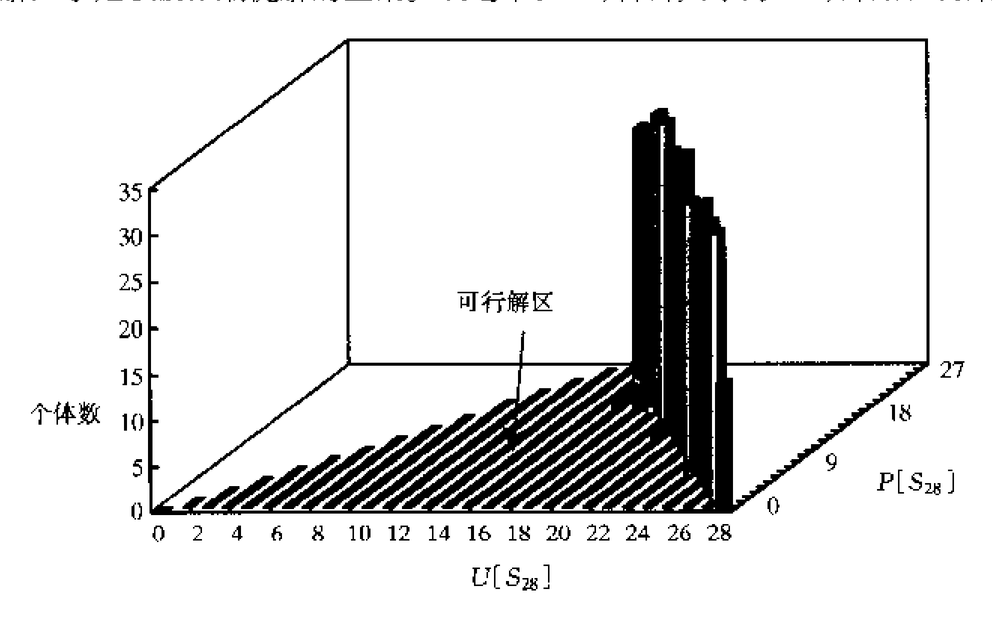


图6.8　NPGA 算法，第200代个体分布

三个个体进行非精确偏好包含，如表 6.5 所示，表中前 3 个个体分别用 x_1，x_2 和 x_3 表示。

表 6.5 初始种群中的个体

NO.	个体	$U[S_{28}]$	$P[S_{28}]$	$v(U[S_{28}])$	$v(P[S_{28}])$
1	1011101101010101110110001001	16	18	0.571	0.667
2	0110010110111011010101111001	17	17	0.607	0.630
3	0010101000000110101101010101	12	20	0.429	0.740
4	1000001110001010110011001101	13	14	0.464	0.518
⋮	⋮	⋮	⋮	⋮	⋮
100	0100111110001010110011001111	16	13	0.571	0.481

如果选择 $x_3 \prec x_1 \prec x_2$，即包含 $P[S_{28}]$ 比 $U[S_{28}]$ 更重要的偏好信息，经过 200 代进化运算，获得的最终代 100 个个体统计如表 6.6 所示，其中 21 个个体的目标值 $P[S_{28}]=20$，$U[S_{28}]=18$；49 个个体目标值 $P[S_{28}]=22$，$U[S_{28}]=17$；30 个个体的目标值 $P[S_{28}]=24$，$U[S_{28}]=16$。显然，作为最终解的个体的 $P[S_{28}]$ 很大。

表 6.6 第 200 代种群个体统计($x_3 \succ x_1 \succ x_2$)

NO.	个体	$U[S_{28}]$	$P[S_{28}]$	$v(U[S_{28}])$	$v(P[S_{28}])$
1	1010101101010110101011101101	17	22	0.607	0.815
2	1101010101011010101110110101				
⋮	⋮				
49	1010101010110101011011101011				
50	1101110101010110101101110101	18	20	0.642	0.740
51	0111011010101110101110101011				
⋮	⋮				
70	1011010101011010111110110101				
71	1010101010101101010110110101	16	24	0.571	0.889
72	1101101010101010101010110101				
⋮	⋮				
100	1010110110101010101010110101				

如果选择 $x_2 \succ x_1 \succ x_3$，即包含 $U[S_{28}]$ 比 $P[S_{28}]$ 更重要的偏好信息，经过 200 代进化运算，获得的最终代 100 个个体统计如表 6.7 所示，其中 8 个个体的目标值 $P[S_{28}]=0$，$U[S_{28}]=28$；92 个个体目标值 $P[S_{28}]=2$，$U[S_{28}]=27$。显然，作为最终解的个体的 $U[S_{28}]$ 很大。

表 6.7 第 200 代种群个体统计($x_2 \succ x_1 \succ x_3$)

NO.	个体	$U[S_{28}]$	$P[S_{28}]$	$v(U[S_{28}])$	$v(P[S_{28}])$
1	1111111111111111111111111111	28	0	1	0
2	1111111111111111111111111111				
⋮	⋮				
8	1111111111111111111111111111				
9	1111111111110111111111111111	27	2	0.964	0.074
10	1110111111111111111111111111				
⋮	⋮				
100	1111101111111111111111111111				

图6.9和图6.10所示，两种情况下最终解都处于Pareto前沿，而且与决策者的偏好非常吻合。实际上是由于遗传进化中保持了决策者非精确偏好信息而演化的结果。

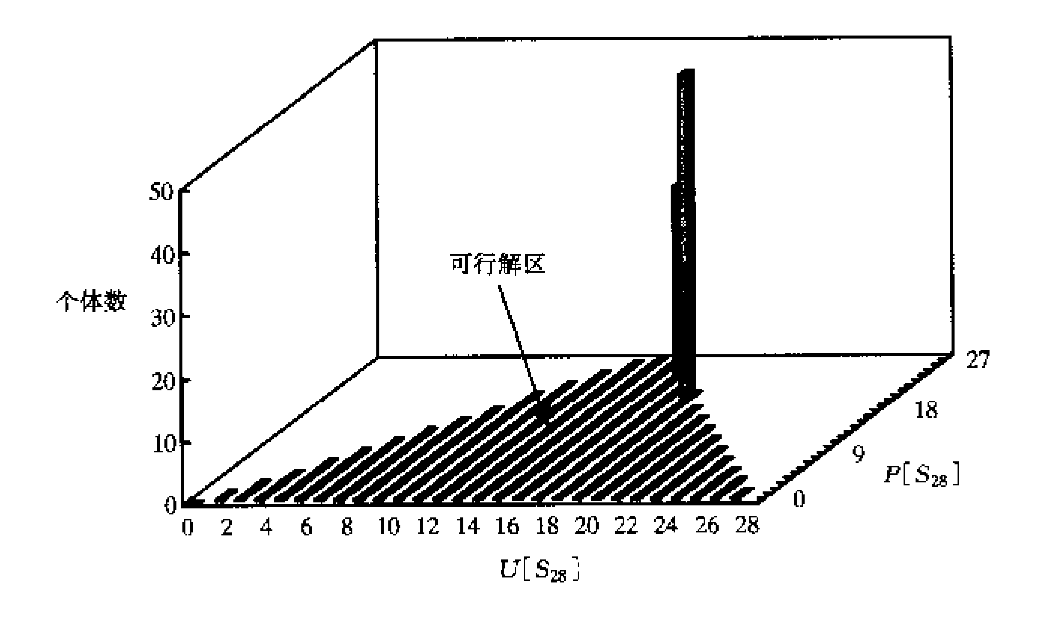


图6.9 第200代个体分布($x_3 \succ x_1 \succ x_2$)

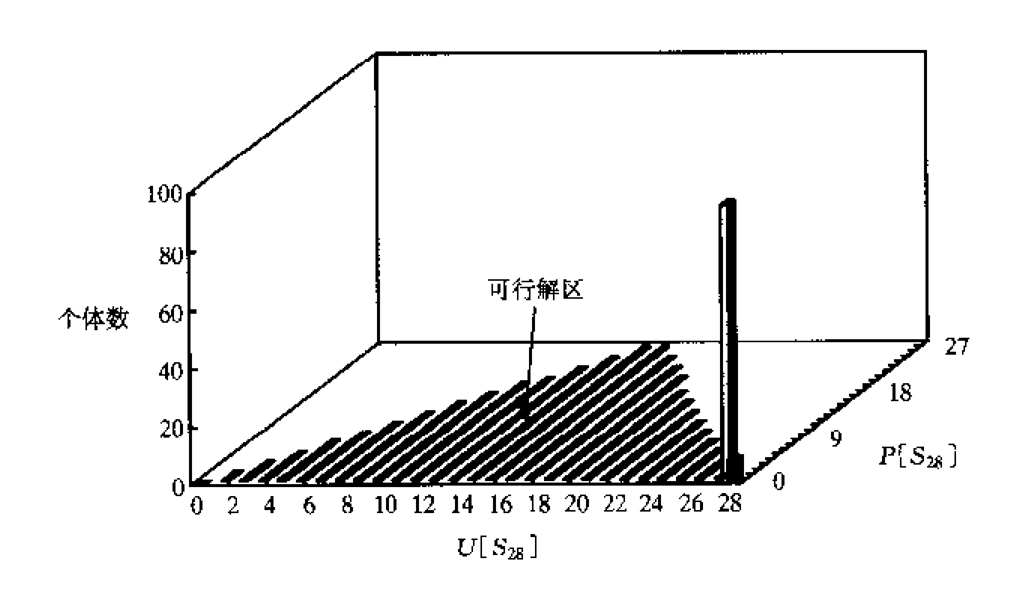


图6.10 第200代个体分布($x_2 \succ x_1 \succ x_3$)

遗传算法的内在并行机制及其全局优化的特点适合于多目标优化问题的解决，特别是目标函数多、数学表达式非线性或者不明确、优化变量多、常规方法难以奏效的复杂场合，如控制系统优化设计。基于Pareto秩的MOGA有利于寻找最优解集，结合决策者的偏好信息构造适应度值，有利于寻求满意解集。本文提出的交互式多目标遗传算法以较容易的方式获得非精确偏好信息，据此世代进化获得与决策者意愿吻合的解集。此外，分享法(Sharing method)对于保持Pareto前沿的遗传多样性往往是必需的，通过减少相似个体的复制量达到同时探索多个区域的目的。

参考文献

[1] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer－Verlag, Second, Extended Edition, 1994

[2] Michalewica Z, Janikow C. Handling Constraints in Genetic Algorithms. In: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1991, 151～157

[3] De Jong K A. An Anlysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral Dissertation, University of Michigan, 1975

[4] Macready W G, Wolpert D H. What Makes an Optimization Problem Hard? SFI－TR－95－05－046, the Santa Fe Institute, 1995

[5] Houck C R, Joines J A. A Genetic Algorithm for Function Optimization: A MATLAB Implementation. NCSU－IE TR95－09, 1995

[6] Myung H, Kim J H. Lagrangian－Based Evolutionary Programming for Constrained Optimization. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 35～44

[7] Pan Z, Kang L. An Adaptive Evolutionary Algorithm for Numerical Optimization. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 27～34

[8] Krishnakumar K. Solving Large Parameter Optimization Problems Using A Genetic Algorithm with Stochastic Coding. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed.), Wiley, 1995, 287～301

[9] Smith A, Tate D. Genetic Optimization Using a Penalty Function, Proceedings of 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1993, 499～503

[10] Beasley D, Bull D R. A Sequencing Niche Technique for Multimodal Function Optimization. Evolutionary Computation, 1993, 1(2): 101～125

[11] 徐光辉. 运筹学基础手册. 北京: 科学出版社, 1999

[12] 丁承民, 张传生等. 正交试验遗传算法及其在函数优化中的应用. 系统工程与电子技术, 1997(10): 57～60

[13] 樊叔维, 汪洁等. 遗传算法在电力变压器和电机全局优化设计中的应用研究. 西安交通大学学报, 1996, 30(6): 15～21

[14] 苟先太, 金炜东. 有约束优化中遗传算法的应用. 西南交通大学学报, 1997, 32(4): 433～437

[15] 鹿跃丽, 周力平. 遗传算法用于工程结构优化设计的研究. 郑州工业大学学报, 1998, 19(2): 35～39

[16] 侯格贤, 吴成柯. 遗传算法的性能分析. 控制与决策, 1999, 14(3): 257～260

[17] 刘铁南, 陈广义, 刘延力. 模拟生物种族形成的进化算法与多峰函数优化. 控制与决策, 1999, 14(3): 185～188

第7章 遗传算法与组合最优化

上一章中，我们讨论了遗传算法在连续函数最优化问题中的应用。本章主要考察遗传算法在组合最优化问题中的几个应用。

首先，7.1节中将介绍著名的巡回旅行商问题(Traveling Salesman Problem, TSP)，它属于NP完全问题，给定一组 n 个城市和它们两两之间的直达距离，寻找一条闭合的旅程，使得每个城市刚好经过一次且总的旅行距离最短。用遗传算法解决TSP，一个旅程很自然地表示为 n 个城市的排列，但基于二进制编码的交叉和变异操作不能适用，可以设计一组被称为重排的新的操作来处理这类表示问题。然后，我们探讨作业调度问题(Job shop Scheduling Problem, JSP)，也是一类NP难题。车间作业是指利用车间资源对某一对象进行生产的过程，作业调度实际上是对车间作业进行有效排序，使某个目标函数最小。例如，一台机床有多个工具，加工多种工件，各工件有确定的加工期限约束，需要制定一个月的加工计划保证完成所有的加工任务。由于JSP具有离散、动态、多变量耦合等属性，应用遗传算法具有一定的难度，主要体现在遗传编码方法上。7.2节将结合实例讨论这方面的应用。最后，作为组合最优化问题的另一个典型实例——背包问题(knapsack problem)，在一定重量限制下，在背包中分别存放多种重量、价值不同的物件，如何进行物件组合的选择，使背包内物件总价值最大。遗传算法应用于背包问题涉及到约束条件满足下的遗传编码方法，以及交叉、变异操作算子的设计问题。我们将通过实例分析，考察遗传算法应用于组合优化问题上的搜索能力。

7.1 巡回旅行商问题

巡回旅行商问题(TSP)，也称为货郎担问题，是一个较古老的问题。最早可以追溯到1759年Euler提出的骑士旅行问题。1948年，由美国兰德公司推动，TSP成为近代组合优化领域的一个典型难题。应该说，TSP是一个具有广泛的应用背景和重要理论价值的组合优化问题，它已经被证明属于NP难题。

用图语言来描述TSP，给出一个图 $G=(V,E)$，每边 $e\in E$ 上有非负权值 $w(e)$，寻找 G 的Hamilton圈 C，使得 C 的总权 $W(C)=\sum_{e\in E(C)} w(e)$ 最小。

几十年来，出现了很多近似优化算法，如近邻法(nearest neighbor)、贪心算法(greedy algorithm)、最近插入法(nearest insertion)、最远插入法(farthest insertion)、双极小生成树法(double minimum spanning tree)等等。近年来，有很多解决该问题的较为有效的算法不断被推出，例如Hopfield神经网络方法、模拟退火方法以及遗传算法方法。

TSP搜索空间随着城市数 n 的增加而增大，所有的旅程路线组合数为 $(n-1)!/2$。5个城市的情形对应 $120/10=12$ 条路线，10个城市的情形对应 $3\ 628\ 800/20=181\ 440$ 条路线，

100个城市的情形则对应有4.666 3×10^155条路线。在如此庞大的搜索空间中寻求最优解,对于常规方法和现有的计算工具而言,存在着诸多的计算困难。借助遗传算法的搜索能力解决TSP问题,是很自然的想法。但如果将一条旅程路线表示为一个 n 城市的排列,基于二进制编码的交叉和变异操作就不能适用,所以需要重新设计遗传操作,以适应这类遗传基因表示问题。下面我们结合9个城市的TSP,讨论几种遗传算法的应用方法。

7.1.1 顺序表示与交叉

1985年,Grefenstette等针对TSP提出了基于顺序表示(ordinal representation)的遗传基因编码方法。顺序表示是指将所有城市依次排列构成一个顺序表(order list),对于一条旅程,可以依旅行经过顺序处理每个城市,每个城市在顺序表中的顺序就是一个遗传因子的表示,每次处理完一个城市,从顺序表中去掉该城市。处理完所有城市后,将每个城市的遗传因子表示连接起来,即成为一条旅程的基因表示(染色体编码)。例如,顺序表 $C=(1,2,3,4,5,6,7,8,9)$,一条旅程为1-2-4-3-8-5-9-6-7。按照这种编码方法,这条旅程的编码为表 $l=$ (1 1 2 1 4 1 3 1 1)。

由于采用这种顺序表示技术,可以采用基本遗传算法的交叉操作,例如,单点交叉的情形,父个体1和父个体2分别为:

p_1: (1 1 2 1 | 1 4 1 3 1 1)

p_2: (5 1 5 5 | 5 3 3 3 2 1)

它们代表旅程分别为:

1-2-4-3-8-5-9-6-7

5-1-7-8-9-4-6-3-2

两个个体在交叉点处进行基因重组,生成子个体1和子个体2,分别为:

o_1: (1 1 2 1 5 3 3 2 1)

o_2: (5 1 5 5 4 1 3 1 1)

它们代表旅程分别为:

1-2-4-3-9-7-8-6-5

5-1-7-8-6-2-9-3-4

从上述单点交叉来看,交叉点右侧部分的旅程发生了随机变化,但交叉点左侧部分的旅程未发生改变,由于顺序表示和单点交叉存在这个缺点,这种方法的适用性存在一定的问题。

7.1.2 路径表示与交叉

路径表示(path representation)是表示旅程对应的基因编码的最自然、最简单的表示方法。例如,旅程(5-1-7-8-9-4-6-2-3可以直接表示为(5 1 7 8 9 4 6 2 3),基于路径表示的编码方法,要求一个个体(即一条旅程)的染色体编码中不允许有重复的基因码,也就是说要满足任一个城市必须而且只能访问一次的约束。这样,基本遗传算法的交叉操作生成的个体一般不能满足这一约束条件。为此,人们提出了一组称为重排操作的新的操作来处理这类表示问题,它包括三种操作:部分匹配交叉(Partially Matched Crossover, PMX)、顺序交叉(Ordered Crossover, OX)、循环交叉(Cycle Crossover, CX)。

(1) **部分匹配交叉** 1985年,Goldberg等针对TSP提出了基于路径表示的部分匹配交叉(PMX)操作,PMX操作要求随机选取的两个交叉点,以便确定一个匹配段,根据两个父个体

中两个交叉点之间的中间段给出的映射关系生成两个子个体。

例如，对下面两个父个体的表示，随机地选择两个交叉点“｜”。

$$p_1:\quad (1\,2\,3 \mid 4\,5\,6\,7 \mid 8\,9)$$

$$p_2:\quad (4\,5\,2 \mid 1\,8\,7\,6 \mid 9\,3)$$

首先，两个交叉点之间的中间段交换，得到：

$$o_1:\quad (\mathrm{x\,x\,x} \mid 1\,8\,7\,6 \mid \mathrm{x\,x})$$

$$o_2:\quad (\mathrm{x\,x\,x} \mid 4\,5\,6\,7 \mid \mathrm{x\,x})$$

其中 x 表示暂未定义码(本节下同)，得到中间段的映射关系，有：

$$1\longleftrightarrow 4,\ 8\longleftrightarrow 5,\ 7\longleftrightarrow 6,\ 6\longleftrightarrow 7$$

然后，对子个体 1、子个体 2 中 x 部分，分别保留从其父个体中继承未选定城市码 2, 3, 9，得到：

$$o_1:\quad (\mathrm{x}\,2\,3 \mid 1\,8\,7\,6 \mid \mathrm{x}\,9)$$

$$o_2:\quad (\mathrm{x\,x}\,2 \mid 4\,5\,6\,7 \mid 9\,3)$$

最后，根据中间段的映射关系，对于上面子个体 1 的第一个 x，使用最初父码 1，由 1←→4 交换得到第一个 x 为 4，类似地子个体 2 的第二个 x，使用最初父码 8，由 8←→5 交换得到子个体 1 的第二个 x 为 5。如果映射关系中存在传递关系，即备选交换有多个码，则选择此前未确定的一个码作为交换。类似地进行操作，最终得到的子个体为：

$$o_1:\quad (4\,2\,3 \mid 1\,8\,7\,6 \mid 5\,9)$$

$$o_2:\quad (1\,8\,2 \mid 4\,5\,6\,7 \mid 9\,3)$$

(2) **顺序交叉**　1985 年，Davis 等针对 TSP 提出了基于路径表示的顺序交叉(OX)操作。OX 操作能保留排列并融合不同排列的有序结构单元。两个父个体交叉时，通过选择父个体 1 的一部分，保存父个体 2 中城市码的相对顺序生成子个体。

例如，对下面两个父个体的表示，与 PMX 操作一样随机选择两个交叉点“｜”。

$$p_1:\quad (1\,2\,3 \mid 4\,5\,6\,7 \mid 8\,9)$$

$$p_2:\quad (4\,5\,2 \mid 1\,8\,7\,6 \mid 9\,3)$$

首先，两个交叉点之间的中间段保存不变，得到：

$$o_1:\quad (\mathrm{x\,x\,x} \mid 4\,5\,6\,7 \mid \mathrm{x\,x})$$

$$o_2:\quad (\mathrm{x\,x\,x} \mid 1\,8\,7\,6 \mid \mathrm{x\,x})$$

然后，记取父个体 2 从第二个交叉点开始城市码的排列顺序，当到达表尾时，返回表头继续记录城市码，直至到达第二个交叉点结束，这样便获得了父个体 2 从第二个交叉点开始的城市码排列顺序为 9－3－4－5－2－1－8－7－6。对于父个体 1 而言，已有城市码有 4, 5, 6, 7，将它们从父个体 2 的城市码排列顺序中去掉，得到排列顺序 9－3－2－1－8，再将这个排列顺序复制给父个体 1，复制的起点也是从第二个交叉点开始，以此决定子个体 1 对应位置的未知码 x，这样子个体 1 生成为：

$$o_1:\quad (2\,1\,8\,4\,5\,6\,7\,9\,3)$$

同样，可以产生子个体 2 为：

$$o_2:\quad (3\,4\,5\,1\,8\,7\,6\,9\,2)$$

(3) **循环交叉**　1987 年，Oliver 等针对 TSP 提出了循环交叉(CX)操作。CX 操作中子个

体中的城市码顺序根据任一父个体产生。

例如，下面为两个父个体：

p_1: (1 2 3 4 5 6 7 8 9)

p_2: (4 1 2 8 7 6 9 3 5)

先从父个体 1 中取第一个城市码，生成子个体 1：

o_1: (1 x x x x x x x x)

子个体的所有城市从任一父个体的相同位置取出，由于父个体 2 中与父个体 1 的第 1 个城市码 1 对应的城市码为 4，因此决定子个体 1 的第 4 个城市码为 4：

o_1: (1 x x 4 x x x x x)

接下来，选出的城市码为 4 对应父个体 2 的城市码为 8，因此，确定子个体 1 的第 8 个城市码：

o_1: (1 x x 4 x x x 8 x)

按照这样的规则，依次确定子个体的 1 城市码，直到又再次选择城市 1，称为一个循环，获得：

o_1: (1 2 3 4 x x x 8 x)

最后，将子个体 1 中未定义码 x 用父个体 2 中的对应城市码，完成子个体 1 的产生过程。有：

o_1: (1 2 3 4 7 6 9 8 5)

同样，可以产生子个体 2 为：

o_2: (4 1 2 8 5 6 7 3 9)

(4) **边重组** 上述几种 TSP 交叉操作基本上考虑的是城市的位置和顺序，未考虑城市间的连接。Grefenstette 认为遗传算法应用于 TSP，其遗传操作不仅要考虑城市的位置，而且有必要考虑城市间的关系，城市间的关系定义为边(edge)，让子个体继承父个体中边的信息，设计围绕边的遗传操作很有意义。1989 年，Whitley 等提出了一种被称为边重组(Edge Recombination, ER)交叉操作，使子个体能够从父个体继承 95% ～99%的边的信息。ER 操作根据继承两个父个体定义的旅程中城市间的相邻状况生成子个体。

例如，一条旅程(3 1 2 8 7 4 6 9 5)中可以定义边有(3 1)，(1 2)，(2 8)，(8 7)，(7 4)，(4 6)，(6 9)，(9 5)，(5 3)。TSP 中，最小化的目标函数是由合法旅程中所有边的总和构成的，从这个意义上讲，旅程中城市的位置不是特别重要的。此外，边(1 3)与边(3 1)表示城市 1 与城市 3 的邻接关系，边的方向也不是重要的。因此，ER 操作中，对于每一城市，将任意一父个体中与之邻接的其他所有城市列出构成边表，然后利用两个父个体对应的旅程中的边表来设计交叉操作。

考虑下面两个父个体表示：

p_1: (1 2 3 4 5 6 7 8 9)

p_2: (4 1 2 8 7 6 9 3 5)

每一城市的边表为：

城市 1：通向其他城市的边有 9 2 4；

城市 2：通向其他城市的边有 1 3 8；

城市 3：通向其他城市的边有 2 4 9 5；

城市 4：通向其他城市的边有 3 5 1；

城市 5：通向其他城市的边有 4 6 3；

城市 6：通向其他城市的边有 5 7 9；

城市 7：通向其他城市的边有 6 8；

城市 8：通向其他城市的边有 7 9 2；

城市 9：通向其他城市的边有 8 1 6 3。

子个体的生成从子个体的第一个城市码选择开始，可以随机地选择两个父个体的第一个城市码(城市 1 和城市 4)。已选城市的边表中所有城市对应边表中边数最少的城市码作为下一个城市码，如果存在边数相等的边表，可以从中随机选择城市码。这样的选择过程反复进行直至生成子个体所有城市码。假定子个体首先随机选定城市 1，城市 1 与另三个城市 9，2，4 邻接，从中选择一个城市作为下一个城市码，这里，城市 9 的边表长为 4，城市 2 和城市 4 的边表长均为 3，于是，从城市 2 和城市 4 中随机选择一个城市码。假定选择了 4 作为第二个城市码，第三个城市码从城市 4 的边表中所列城市 3 和城市 5 中产生，由于城市 3 的边表长为 4，城市 5 的边表长为 3，因此，选择城市 5。如此进行下去，可以获得一个子个体：

(1 4 5 6 7 8 2 3 9)

细心的读者不难发现，ER 操作可能存在边失败(edge failure)的问题，即中间选定城市码对应的边表所列城市，在该步选择之间已经都进入了选择，接下来无从选择下一个城市码。Whitley 等在一组试验中发现，出现这样的问题概率很小，大致为 1% ~1.5%。为此，1991 年，Starkweather 等提出了一种改进方法，在 ER 操作中不再保留父个体中共同部分的序列。例如，一个边表包含 3 条边：

城市 4：通向其他城市的边有 3 5 1

其中一条边已重复，即(4 5)，在其两个父个体中均有这条边，而(4 3)和(4 1)仅在一个父个体中有定义，因此，将边表中属于重复的边打上标记“ - ”。在上面例子中，新的边表为：

城市 1：通向其他城市的边有 9 -2 4；

城市 2：通向其他城市的边有 -1 3 8；

城市 3：通向其他城市的边有 2 4 9 5；

城市 4：通向其他城市的边有 3 -5 1；

城市 5：通向其他城市的边有 -4 6 3；

城市 6：通向其他城市的边有 5 -7 9；

城市 7：通向其他城市的边有 -6 -8；

城市 8：通向其他城市的边有 -7 9 2；

城市 9：通向其他城市的边有 8 1 6 3。

根据新的边表，ER 操作中产生子个体时优先考虑选择打有标记的城市码。计算结果表明这种处理比含随机选择的处理的性能有相当的改善。

7.1.3　布尔矩阵表示

以上介绍了 TSP 顺序表示和路径表示方法及其遗传操作，这两种表示方法从根本上属于遗传基因码的向量式表示，是否充分反映了一条旅程包含的遗传信息是令人怀疑的。采用非向量表示方法值得研究。1992 年，Fox 和 McMahon 等提出了旅程的矩阵表示方法。他们将一个旅程定义为一个优先权布尔矩阵 $\boldsymbol{M}$，当且仅当城市 i 排在城市 j 之前时矩阵元素 $m_{ij}=1$。例如，一条旅程(3 1 2 8 7 4 6 9 5)可以用如表 7.1 所示的矩阵表示。

这种方法用 $n\times n$ 矩阵 $\boldsymbol{M}$ 代表一条旅程，$\boldsymbol{M}$ 具有如下三个性质：

① 矩阵中 1 的数目为 $n(n-1)/2$;

② $m_{ii}=0, i=1,\cdots,n$;

③ 若 $m_{ij}=1$，且 $m_{jk}=1$，则 $m_{ik}=1$。

针对矩阵表示方法，他们设计了两种类似于二进制交叉算子：交(intersection)和并(union)，在继承父代优良特性的同时保证满足约束要求。

交算子是基于这样的观察：两个矩阵中位进行交运算产生的矩阵即可获得一个合法的旅程矩阵，很明显它满足旅程矩阵的三个性质。例如，两个父个体 $p_1=(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ 和 $p_2=(4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$ 由两个矩阵表示，分别为表 7.2 和表 7.3。对这两个矩阵求交得到的矩阵如表 7.4 所示。由交结果强加的部分次序为：城市 1 优先于城市 2, 3, 5, 6, 7, 8, 9；城市 2 优先于城市 3, 5, 6, 7, 8, 9；城市 3 优先于城市 5；城市 4 优先于城市 5, 6, 7, 8, 9；城市 6, 7, 8 优先于城市 9。交算子的下一个步骤是选择一个父个体，将该父个体矩阵中特有的"1"加入到交的第一阶段结果矩阵中，并通过分析行列之和完成一个序列。表 7.5 为交的最后一个可能结果矩阵，它表示旅程(1 2 4 8 7 6 3 5 9)。

表 7.1 一条旅程的矩阵表示

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1
3	1	1	0	1	1	1	1	1	1
4	0	0	0	0	1	1	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1
7	0	0	0	1	1	1	0	0	1
8	0	0	0	1	1	1	1	0	1
9	0	0	0	0	1	0	0	0	0

表 7.2 p_1 的矩阵表示

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	1	1	1	1	1	1
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

表 7.3 p_2 的矩阵表示

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	1	1	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	1	0	1	0	0	0	0

表 7.4 交算子的第一阶段

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	1	0	1	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

表 7.5 交算子的最后结果

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	0	0	0	1
4	0	0	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	1
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	0	0	0	0	0	0	0

并算子是基于这样的观察：一个矩阵的位子集可以安全地与另一个矩阵的位子集合并，只要这两个子集的交为空。该算子将城市集合分割成两个分离的组，第一组城市从第一个矩阵拷贝位；第二个城市从第二个矩阵拷贝位。最后，通过分析行和列的和将矩阵变为一个序列。例如两个父个体 p_1 和 p_2 分割成{1, 2, 3, 4}和{5, 6, 7, 8, 9}，则产生的矩阵如表 7.6 所示。

表 7.6　并算子的第一阶段

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	x	x	x	x	x
2	0	0	1	1	x	x	x	x	x
3	0	0	0	1	x	x	x	x	x
4	0	0	0	0	x	x	x	x	x
5	x	x	x	x	0	0	0	0	0
6	x	x	x	x	1	0	0	0	1
7	x	x	x	x	1	1	0	0	1
8	x	x	x	x	1	1	1	0	1
9	x	x	x	x	1	0	0	0	0

表 7.7　20 城市 TSP 的坐标

城市	x	y	城市	x	y
A	5.294	1.558	K	4.399	1.194
B	4.286	3.622	L	4.660	2.949
C	4.719	2.774	M	1.232	6.440
D	4.185	2.230	N	5.036	0.244
E	0.915	3.821	O	2.710	3.140
F	4.771	6.041	P	1.072	3.454
G	1.524	2.871	Q	5.855	6.203
H	3.447	2.111	R	0.194	1.862
I	3.718	3.665	S	1.762	2.693
J	2.649	2.556	T	2.682	6.097

7.1.4　TSP 的应用示例

表 7.7 列出了一个 20 城市 TSP 中的城市坐标信息。对这一 TSP 实例，应用启发式搜索 A^* 算法，获得最优旅程为 ACLBIQFTMEPRGSOJHDKN，旅程长度为 24.38，如图 7.1 所示。该最优解在展开 17 222 个节点后被找到。应用模拟退火方法得到同样的最优解，从随机初始旅程出发的 1 000 次运行中，模拟退火法找到最优旅程的情况占 79.2%。采用遗传算法方法，应用选择操作和 OX 操作，种群大小为 300，交叉概率为 0.45，终止代数 300，共运行 500 次，得到同样的最优解，在所有 500 次运行中均被找到。

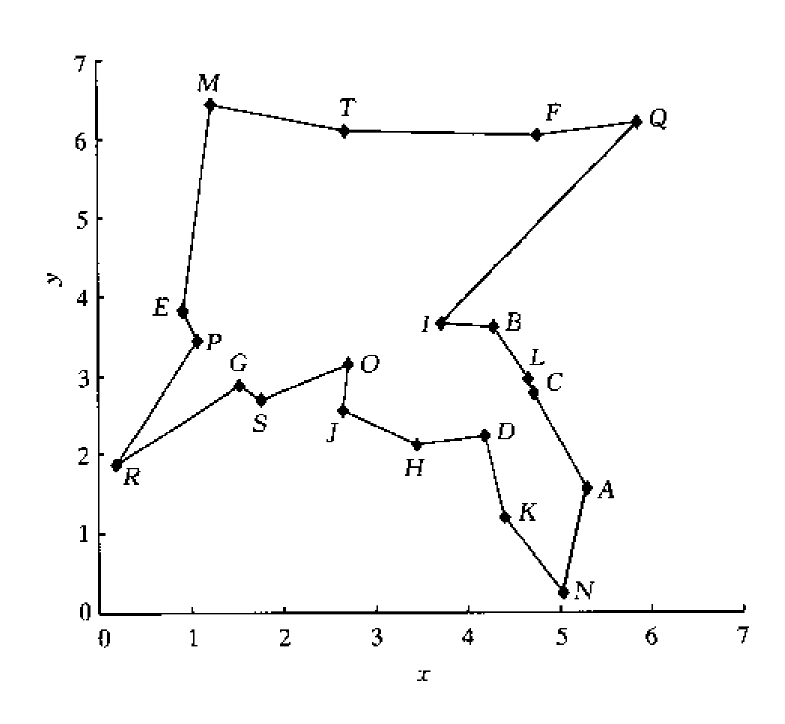


图 7.1　20 城市 TSP 的最优解

7.2 作业调度问题

作业调度问题(Job shop Scheduling Problem, JSP)是一种资源分配问题。这里的资源主要是指设备资源,问题的求解目标是要找到一个将一组工件安排到设备上去,以使作业可为"最优"完成的方案。每个作业可由一些任务组成,而每个任务必须由特定的设备处理。一个调度是按先后顺序条件将所有任务安排到设备上的一种方案。通常,约束的数目很大,使 JSP 成为一个非常难解的组合问题(NP 完全问题)。物流调度问题(Flow shop Scheduling Problem, FSP)则是具有更严格条件的 JSP 特例,并可约化为旅行商问题(TSP)。已经有许多用于求解 JSP 的最优化方法被提出,包括分枝定界法、动态规划法、拉格朗日松弛法和神经网络映射算法等。但由于问题本身难度很大,多数现有的最优化算法只适用于规模较小的问题,另一方面,许多工业界依靠经验或计算机模拟生成可行调度,这样得到的可行调度不能保证最好的性能。从根本上说,评价调度算法的性能很困难,因为获取规模很大且最优调度已知的测试问题本身就很困难。

遗传算法在作业调度上的应用,是近年来才发展起来的研究方向,对复杂工业过程的建模、控制和优化领域的研究有十分重要的意义。Florida 大学的 J. E. Biegel 和 J. J. Davern 最早于 1992 年提出了"用于车间作业调度的遗传算法",浙江大学纪树新于 1995 年发表了题为"基于遗传算法的车间作业调度系统研究"的博士论文。本节以一个金属加工车间的较简单作业调度问题为例,探讨遗传算法的应用。

7.2.1 柔性调度系统(Flexible Scheduling System)

假定一台机器用于金属加工,该机器配备多个加工刀具,加工多种工件,各工件有确定的加工期限约束,需要制定一个月的加工计划保证完成所有的加工任务。工件的加工过程如图 7.2所示。

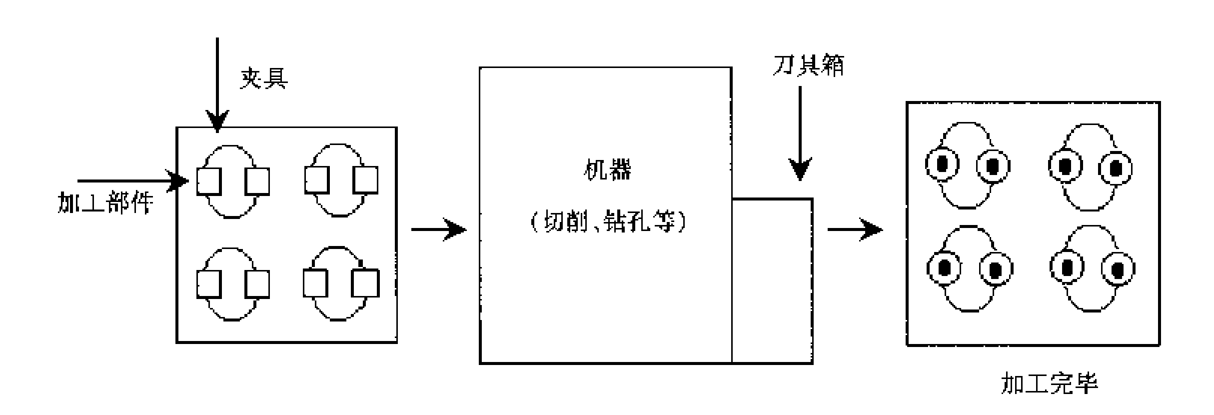


图 7.2 一个金工车间加工过程示意图

这里,假定工件安装在夹具上,每个工件一个夹具,并且一天内能够使用的夹具数目确定。工件安装在夹具上按工序依次在一台机器上完成切削、钻孔等加工任务,与加工任务相对应,可以在刀具箱中自动选择、交换合适的加工刀具。而且,对于不同的工件完成某项加工任务,可以选择通用刀具。

据加工车间数据统计和修正,一个月的加工计划数据如表 7.8 所示。以工件 01 为例,表

中的数据可作如下解释：要求交工期为第14天，完成50个数量的工件，1个夹具可同时安装2个工件，且该夹具一天内只能使用2个，每个工件需要0.650 d时间。

根据表7.8，可以获得作业调度方面的一些数据，如表7.9所示。表中对于每一个工件，1个夹具加工时间＝1工件加工时间×每个夹具可同时加工工件数；总加工时间＝1个夹具加工时间×加工数/每个夹具同时加工工件数；总夹具数＝加工数/每个夹具可同时加工工件数；加工优先评价值＝加工数/(一日可能使用夹具数×每个夹具可同时加工工件数×交工期)；加工优先级按所有工件的加工优先评价值的大小排序得到。

表7.8　一个月加工计划数据表

工件	交工期(d)	加工数目	一天可能使用夹具数	每个夹具可同时加工工件数	加工时间(d)
工件01	14	50	2	2	0.650
工件02	23	50	4	2	1.050
工件03	17	24	2	2	0.200
工件04	11	10	2	2	0.900
工件05	15	26	2	2	0.700
工件06	17	40	1	4	0.550
工件07	12	4	2	2	0.600
工件08	20	40	2	4	0.500
工件09	8	12	2	4	0.525
工件10	5	12	1	4	0.450

表7.9　一个月作业调度数据表

工件	1个夹具加工时间(d)	总加工时间(d)	总夹具数	加工优先评价值	优先级
工件01	1.300	32.500	25	0.893	1
工件02	2.100	52.500	25	0.272	6
工件03	0.400	4.800	12	0.353	5
工件04	1.800	9.000	5	0.227	8
工件05	1.400	18.200	13	0.433	4
工件06	2.200	22.000	10	0.588	3
工件07	1.200	2.400	2	0.083	10
工件08	2.000	20.000	10	0.250	7
工件09	2.100	6.300	3	0.180	9
工件10	1.800	5.400	3	0.600	2

表7.10列出了使用刀具个数的数据。表中工件按优先级顺序排列。一天内准备加工的工件中优先级高的工件的列中，将准备加工工件的使用刀具个数加起来，便可得到该天使用刀具总数。例如，某天内加工工件05，01，02的情况下，先取出优先级高的工件05所在的列，将该列中工件05的刀具数24、工件01的刀具数15和工件02的刀具数10合计，得到该天使用总刀具数为49。

考虑到机器的加工能力和车间作业的实际，一天内可以使用总刀具数限定为99个。要求机器一天内所有工件累计加工时间在11天以内。

表7.10 使用刀具数据表

	工件05	工件02	工件06	工件04	工件03	工件01	工件07	工件08	工件09	工件10
工件05	24									
工件02	10	24								
工件06	14	12	33							
工件04	8	8	10	34						
工件03	11	6	15	4	20					
工件01	15	10	7	4	6	27				
工件07	5	6	6	3	13	12	19			
工件08	14	7	10	7	15	13	6	25		
工件09	7	9	5	3	6	2	7	8	10	
工件10	4	6	7	3	6	10	11	4	24	5

7.2.2 JSP的数学模型及其参变量

对于上述JSP，按最小化各工件交工延迟和最小化总加工时间的目的设计目标函数，再考虑用加工计划表构成约束条件，建立相应的数学模型。

首先，定义以下符号和变量：

i：工件号；

j：从加工开始的天数计数；

y_{ij}：工件 i 在第 j 天的加工时间；

b_{ij}：工件 i 在第 j 天的使用夹具数目；

c_{ij}：工件 i 在第 j 天的使用刀具数目；

t_i：工件 i 的总加工时间；

$\bar{b}_i$：工件 i 可能使用的夹具数；

s_i：工件 i 的交工时间；

X_i：工件 i 加工日的集合。

根据前面分析，JSP的目标函数为

$$\text{Minimize} \quad f = \sum_{i=1}^{10}\sum_{j\in X_i}(s_i - j)^2 \tag{7.1}$$

式(7.1)既考虑了总加工时间的最小化，使工件加工时间集中紧凑，又可保证各工件交工期的总延误最小。

为了实现工件调度作业系统的柔性化，存在一个需要反映决策者偏好的问题。决策者可以选择工件最大完工时间最小化，也可以选择工件最大延迟时间最小化，这两个目标往往是相互矛盾的。为此，1997年森田裕之和加藤直树提出了这一问题解决的多目标遗传算法。

这里，我们仍按单目标优化问题处理。对于每个工件引入一个偏好参数 p_i 来处理。p_i

表示工件 i 在加工期内加工滞后的程度($0\leqslant p_i\leqslant 1$)：$p_i=0$ 表示工件 i 尽可能在初期完工；$p_i=0.5$ 表示工件 i 尽可能在中期完工；$p_i=1$ 表示工件 i 尽可能在接近交工期时完工。因此，新的目标函数定义为：

$$\text{Minimize}\quad f=\sum_{i=1}^{10}\sum_{j\in K_i}(\text{Int}(p_i\times x_i)-j)^2 \tag{7.2}$$

该问题应满足以下约束条件：

① 加工时间约束，要求机器一天内所有工件累计加工时间在11日以内。

$$\sum_{i=1}^{10}y_{ij}\leqslant 11,\qquad j=1,\cdots,23 \tag{7.3}$$

② 刀具个数约束，要求一天内使用刀具总数在99个以内。

$$\sum_{i=1}^{10}c_{ij}\leqslant 99,\qquad j=1,\cdots,23 \tag{7.4}$$

③ 交工期约束，要求所有工件在交工期前加工完毕。

$$\sum_{j=1}^{s_i}y_{ij}=t_i,\qquad i=1,\cdots,10 \tag{7.5}$$

④ 有关夹具约束

$$b_{ij}\leqslant b_i,\qquad i=1,\cdots,10,j=1,\cdots,23 \tag{7.6}$$

7.2.3 JSP的遗传算法应用

遗传算法应用于上述问题，求解作业调度的近似最优解。下面我们先逐一讨论遗传算法设计的几个主要问题，然后给出上述实例的模拟结果。

1. 编码

遗传算法中个体的染色体表示采用矩阵描述，$m\times n$ 的矩阵 $\boldsymbol{Y}=[y_{ij}]$，$i=1,\cdots m$，$j=1,\cdots,n$(m 为从加工开始的天数，n 为工件的优先顺序)。y_{ij}表示工件 i 在第 j 日的加工时间。如果将矩阵 $\boldsymbol{Y}$ 作为个体的编码表示，y_{ij}采用随机产生的方法生成时，根据 y_{ij}计算的一天内工件的加工数会出现不完整的情况。因此，考虑将表示工件 i 在第 j 日使用的夹具数 b_{ij}的矩阵 $\boldsymbol{B}=[b_{ij}]$，作为个体的编码表示。为此，约束条件中的加工时间变量需要转换为夹具数变量，加工时间约束(7.3)和交工期约束(7.5)分别变换为以下的描述：

$$\sum_{i=1}^{10}(b_{ij}\times h_i)\leqslant 11,\qquad j=1,\cdots,23 \tag{7.7}$$

$$\sum_{j=1}^{s_i}b_{ij}=r_i,\qquad i=1,\cdots,10 \tag{7.8}$$

上式中，h_i 为工件 i 使用1个夹具的当量加工时间，r_i 为工件 i 使用的总夹具数。

由于决策变量定义为 b_{ij}，所以原优化问题的组成转化为式(7.1)或式(7.2)的目标函数，约束条件为式(7.4)、式(7.6)、式(7.7)和式(7.8)。

2. 初始种群的生成

原则上，初始种群的个体随机产生，并且要求同时满足约束条件式(7.4)、式(7.6)、式(7.7)和式(7.8)。可按以下五个步骤产生一个可行解，作为初始种群的个体。

第1步　令矩阵 $\boldsymbol{B}$ 的所有元素 $b_{ij}=0$；

第2步　矩阵 $\boldsymbol{B}$ 的列方向(即工件号按距交工期富裕时间)按降序排列,依次处理每个工件。

第3步　对于排在第一位的工件,对其所在行的任一元素,对应地在交工期内随机选择一个加工日。在可能使用的夹具数目范围内,添加夹具数目,直到达到该工件总夹具数目为止。然后执行第二位工件的处理。

第4步　对于排在第二位的工件,对其所在行的任一元素,对应地在交工期内随机选择一个加工日。在可能使用的夹具数目范围内并且在加工时间约束条件下,添加夹具数目,直到达到该工件总夹具数目为止。然后执行下一工件的处理。

第5步　当10个工件的调度处理完成时,检查是否满足刀具个数的约束。如果满足的话,返回到第3步。

3. 选择操作

选择操作基于排序选择的方法。首先,对于种群中的个体对应的一个调度,计算其目标函数。并将个体按目标函数值的升序排列。确定一定的淘汰比例(如20%),将较大的目标函数值对应的个体淘汰掉。然后,对剩余的个体集实行排序选择。

4. 交叉操作

如图7.3所示,由于代表个体的矩阵在行和列的方向上均有相应的约束,两个个体在某个

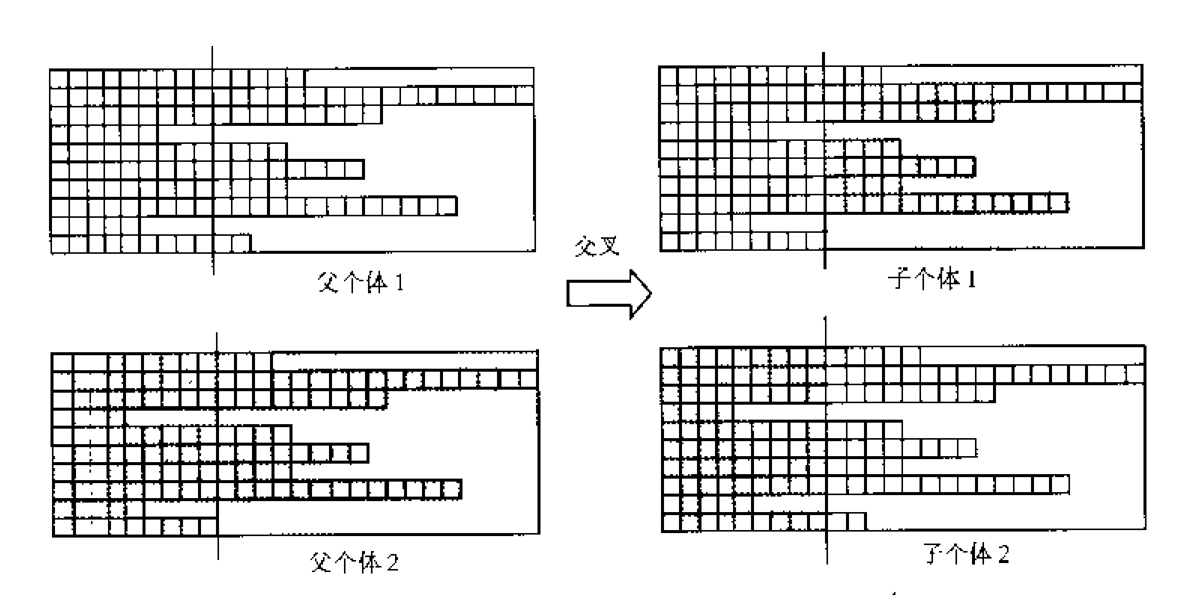


图7.3　交叉操作示意图

随机选择的交叉点实行交叉,生成的子个体很有可能不再满足约束条件。用单点交叉的方法随机选择一个列位置作为交叉点,交叉后生成的子个体满足列方向的约束,但行方向的约束(即夹具数约束)或多或少不能满足。因此,对交叉后的个体修正处理。首先,对夹具数多的工件,在其交工期内随机选择一日,该日内逐次减少夹具,使之达到该工件夹具总数。而对于夹具数少的工件,同样在其交工期内随机选择一日,在考虑列方向约束的同时,添加夹具使之达到该工件夹具总数。如果在未达到总夹具时,按列方向约束不可能再添加夹具的话,这个个体被认为存在致死的遗传基因,应该给予淘汰。在实际模拟中,由于采用修正的处理,大约有5%的个体被淘汰。

5. 变异操作

如图7.4所示,对于一个施加变异操作的个体的矩阵,随机选择两列互换产生新的子个体,

由于这样的操作产生的子个体除交工期约束以外其他约束都能满足，所以，在不能满足交工期约束时，可以再随机选择其他两列来互换产生新的子个体，直至满足交工期约束。

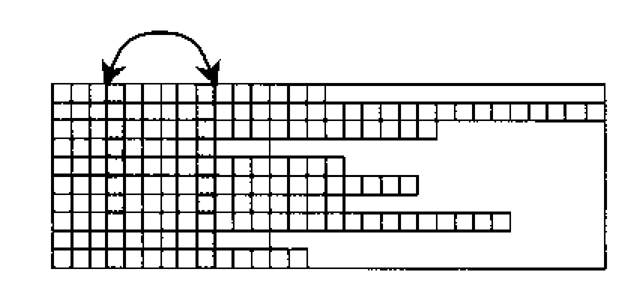


图 7.4　变异操作示意图

6. 模拟结果

对于上述实例的作业调度问题，应用遗传算法，在具体实现和运行时种群规模为 120，最大世代数为 400，交叉率为 0.98，变异率为0.01。经模拟计算，得到该问题的最佳调度如表 7.11 所示，其对应的目标函数值为 1 320。

作为比较，对上述作业调度问题用随机方法计算最佳解。所谓随机法是在满足约束条件的限制下随机产生众多的可行解(如 5 000 个可行解)，然后在其中选择最佳解。表 7.12 对应于一次运行随机法获得的最佳调度，其目标函数值为 4 697。显然，从比较目标函数值可知：遗传算法获得的近似最优解比随机法要好得多；另一方面，从获得的调度表数据分析知，随机法获得的最佳调度中工件的加工时间分布零散，而遗传算法获得的近似最佳解的调度中，工件的加工基本上在接近交工期前完工，这比较符合预期的结果。

表 7.11　遗传算法获得的近似最优调度表

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
工件 01	0	2	2	2	2	2	2	2	2	2	2	2	2	1									
工件 02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	4	3	4	3	4	4
工件 03	0	0	0	0	0	0	0	0	0	0	2	2	0	2	2	2	2						
工件 04	0	0	0	0	0	0	0	2	1	1	1												
工件 05	0	0	0	0	0	0	0	0	2	2	2	2	2	1	2								
工件 06	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1							
工件 07	0	0	0	0	0	0	0	0	2	0	0	0	0										
工件 08	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	2	1	2	1			
工件 09	0	0	0	0	0	2	1	0															
工件 10	0	0	1	1	1																		

表 7.12　随机法获得的最佳调度表

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
工件 01	2	2	1	1	2	2	2	2	2	2	1	2	2	2									
工件 02	0	2	0	0	0	0	0	0	0	2	0	2	0	2	0	2	1	3	1	2	2	3	3
工件 03	0	2	0	1	1	0	2	2	0	1	1	0	0	2	0	0	0						
工件 04	1	0	0	0	0	1	1	0	0	0	2												
工件 05	1	0	0	2	2	0	0	2	1	0	1	2	0	0	2								
工件 06	0	1	0	0	1	1	0	0	1	1	1	0	1	1	1	1	0						
工件 07	0	1	0	0	0	0	1	0	0	0	0	0											
工件 08	0	0	0	1	0	0	0	2	2	0	0	0	0	0	2	0	1	1	0	1			
工件 09	1	0	1	0	0	0	1	0															
工件 10	1	0	0	1	1																		

如果在目标函数中考虑每个工件的决策偏好参数 p_i，按不同的决策偏好进行模拟计算，可以得到不同的最优调度。表 7.13 为工件的决策偏好参数 $p_i=0$ 或 $p_i=1$ 时得到的最优调度，其目标函数值为 1 278。表 7.14 为工件的决策偏好参数 $p_i=0.5$ 或 $p_i=1$ 时得到的最优调度，其目标函数值为 1 005。从调度作业表可以看出，每个工件的加工时间比较集中，其集中时期与其决策偏好参数的取值相关。如果对不同的工件选择不同的偏好参数，可以将工件的加工集中期分散开来，这样有利于缓和加工过程的约束。因此，可以达到柔性处理决策者调度意图的目的，并产生最后可供选择的满意解。

表 7.13　工件的决策偏好参数 $p_i=0$ 或 $p_i=1$ 时，应用遗传算法得到的最优调度

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	p_i
工件 01	0	2	2	2	2	1	2	2	2	2	2	2	2	2										1.0
工件 02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	2	2	3	3	4	4	4	1.0
工件 03	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0							0.0
工件 04	0	0	0	0	0	0	0	0	1	2	2													1.0
工件 05	0	0	0	0	0	0	0	0	2	1	2	2	2	2	2									1.0
工件 06	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0							1.0
工件 07	0	0	0	0	0	0	0	0	0	0	1	1												1.0
工件 08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2				1.0
工件 09	2	1	0	0	0	0	0	0																0.0
工件 10	1	1	1	0	0																			0.0

表 7.14　工件的决策偏好参数 $p_i=0.5$ 或 $p_i=1$ 时，应用遗传算法得到的最优调度

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	p_i
工件 01	0	2	2	2	1	2	2	2	2	2	2	2	2	1										1.0
工件 02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3	3	4	4	4	1.0
工件 03	0	0	0	0	0	2	2	2	2	2	0	2	0	0	0	0	0							0.5
工件 04	0	0	0	0	0	0	0	2	1	1	1													1.0
工件 05	0	0	0	0	0	0	0	0	2	2	2	2	1	2	2									1.0
工件 06	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0							0.5
工件 07	0	0	0	0	0	0	0	0	0	0	1	1												1.0
工件 08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	2	2	2				1.0
工件 09	0	0	0	1	2	0	0	0																0.5
工件 10	0	1	1	1	0																			0.5

7.3　背包问题

本节讨论背包问题(Knapsack problem)，也是一个典型的 NP 完全问题，主要应用于管理中的资源分配、投资决策、装载问题的建模。其求解主要依靠一些启发式算法(如贪心算法)，也可用遗传算法求解。遗传算法应用于背包问题，涉及到约束条件满足下的遗传编码方法，以及交叉、变异操作算子的设计等方面。

背包问题的数学模型实际上是一个 0－1 规划问题。假设有 n 个物件，其重量用 a_j 表示，价值为 $c_j(j=1,\cdots,n)$，背包的最大容纳重量为 b。如果物件 j 被选入背包时，定义变量 $x_j=1$，否则 $x_j=0$。考虑 n 个物件的选择与否，背包内物件的总重量为 $\sum_{j=1}^{n} a_j x_j$，物件的总价值为 $\sum_{j=1}^{n} c_j x_j$，如何决定变量 $x_j(j=1,\cdots,n)$的值(即确定一个物件组合)使背包内物件总价值为最大。这个问题解的总组合数有 2^n 个，其数学模型表示如下：

$$\left.\begin{aligned} &\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} \quad \sum_{j=1}^{n} a_j x_j \leqslant b \\ &x_j = 0 \text{ 或 } 1, \quad j = 1,\cdots,n \end{aligned}\right\} \tag{7.9}$$

上式中，c_j，a_j，b 均为正值。

用贪心算法求解这一问题时，先将物品价值密度 c_j/a_j 的值按降序排列，然后依次将物品放入背包内，直至超出背包最大容纳重量为止。用这种方法求解，只能得到近似最优解，不能保证一定能够得到最优解。

7.3.1　用一般编码方法的遗传算法 SGA

遗传算法应用于背包问题时，如果采用通常的二进制编码，一组 0－1 决策变量$\{x_j(j=1,\cdots,n)\}$直接表示 n 为二进制字符串，作为一个个体的遗传基因表示。在这样的表示方法中，若第 j 位基因码为 1 时，则第 j 个物件被选择；若第 j 位基因码为 0 时，则第 j 个物件不被选择。例如，下面为一个 8 变量的背包问题：

$$\left.\begin{aligned} &\text{Maximize} \quad 5x_1 + 10x_2 + 13x_3 + 4x_4 + 3x_5 + 11x_6 + 13x_7 + 10x_8 \\ &\text{subject to} \quad 2x_1 + 5x_2 + 18x_3 + 3x_4 + 2x_5 + 5x_6 + 10x_7 + 4x_8 \leqslant 25 \\ &x_j = 0 \text{ 或 } 1, \quad j = 1,\cdots,n \end{aligned}\right\} \tag{7.10}$$

当产生一个二进制编码的个体 $\langle 1,0,0,1,0,1,0,1 \rangle$，则该个体对应于选择了物件 1，4，6，8。

处理约束条件有两种方法：一种方法是用罚函数法改造目标函数；另一种方法是结合贪心算法改造染色体的解码过程。后者实际上可以看作是一种混合遗传算法。

(1) **罚函数法**　适应度函数的计算用下式：

$$f_i = \begin{cases} \sum_{j=1}^{n} c_j x_j, & \sum_{j=1}^{n} a_j x_j \leqslant b \\ 0, & \sum_{j=1}^{n} a_j x_j > b \end{cases} \tag{7.11}$$

实际上，上述适应度函数基于一个考虑违背约束条件的惩罚处理，当问题规模很大时，尽管方法可行，但搜索的效率很低，甚至很多情况下所得到的结果比贪心算法的结果还差。

(2) **混合遗传算法**　将启发式搜索算法“贪心算法”引入染色体解码过程中，具体做法很简单，对于那些不满足约束的染色体编码对应的个体，优先装入价值密度较大且编码值为 1 的物品，直至背包容量限制装不下为止，并将未装入的物品编码值修正为 0，形成个体新的染色体编码。

7.3.2 二重结构编码的遗传算法

1994年，坂和正男等提出了一种改进编码方法(即二重结构编码)来考虑约束条件的满足问题，并可以提高遗传算法的搜索效率。

二重结构编码方法如图7.5所示。个体染色体表示的二重结构由变量码和附加码两行组成。上行 $s(i)$ 为变量 x_j 的附加码 $s(i)=j$，下行为变量 $x_{s(i)}$ 对应于附加码 $s(i)$ 的值。

附加码	$s(1)$	$s(2)$	…	$s(i)$	…	$s(n)$
变量码	$x_{s(1)}$	$x_{s(2)}$	…	$x_{s(i)}$	…	$x_{s(n)}$

图7.5 二重结构编码

对某个个体编码时，首先按洗牌方式随机产生附加码 $\{s(i),(i=1,\cdots,n)\}$，列于上行；然后随机产生下行的变量码值(0或1)，这样构成一个个体的二重结构编码。

个体解码时，需要考虑约束条件。如图7.6所示，按照从左到右的顺序，依次考虑变量的附加码，即按顺序考虑附加码为 $s(i)$ 的物件，如果处理到某个物件时违背了约束条件，强制使该物件的变量值 $p_{s(i)}$ 为0，反之，该物件的变量值 $p_{s(i)}$ 为1，直到所有物件都处理完为止。

附加码	$s(1)$	$s(2)$	…	$s(i)$	…	$s(n)$
变量解码值	$p_{s(1)}$	$p_{s(2)}$	…	$p_{s(i)}$	…	$p_{s(n)}$

图7.6 二重结构解码

解码算法的步骤如下：

第1步 $i=1, sum=0$。

第2步 若 $x_{s(i)}=0$，则 $p_{s(i)}=0$，执行第4步，否则，执行第3步。

第3步 若 $sum+a_{s(i)}\leqslant b$，则 $p_{s(i)}=1, sum=sum+a_{s(i)}$，否则 $p_{s(i)}=0$。

第4步 $i=i+1$，若 $i\leqslant n$，返回到第3步，否则终止。

例如，对于一个8变量的背包问题，如果随机产生的附加码序列为4,3,8,1,6,2,5,7，则该个体的二重结构编码如图7.7所示。它对应于一个可行解，即选择了序号4,3,1,6的物件。

4	3	8	1	6	2	5	7
1	1	0	1	1	0	0	0

图7.7 一个个体的二重结构编码

对于上述二重结构编码，交叉操作和变异操作的算子需要重新设计。

对于交叉操作，用通常的操作算子，产生新个体的上行附加码会出现重复。如果采用7.1节介绍的部分匹配交叉(PMX)算子则可以很好地解决这个问题。

图7.8所示的是为两个个体 X，Y 经PMX操作产生两个子个体 X^*，Y^* 的情况。

值得注意的是，PMX操作只是针对个体的上行附加码，子个体的下行变量码值仍根据其父个体中附加码与变量码的对应关系来确定。

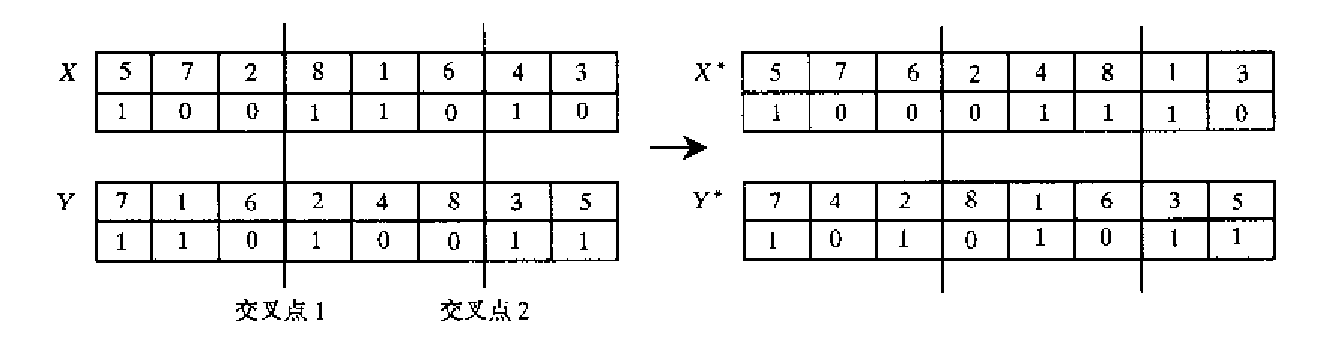


图 7.8　二重结构的 PMX

对于变异操作，采用逆位遗传算子。如图 7.9 所示，对父个体随机选择两个变异点，两点间的上行附加码按相反顺序重新排列，而下行的变量码顺序不变。

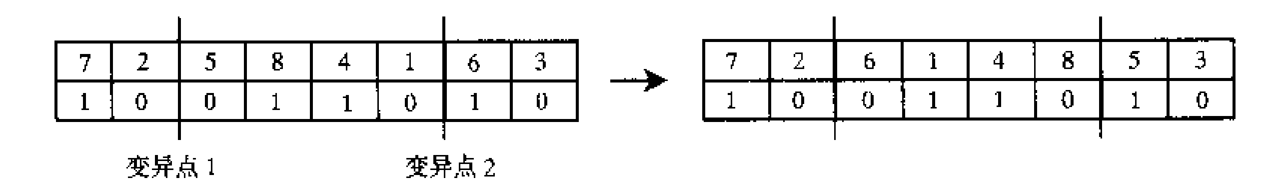


图 7.9　二重结构的逆位变异

7.3.3　模拟结果

下面结合实例将基本遗传算法 SGA 与二重结构编码的遗传算法进行比较。实例中问题规模为 50 个变量，并随机地产生系数值，按以下三个步骤生成一个背包问题，共生成 10 个。

① 50 个系数 c_j，a_j 在区间(0，999]内随机地产生。

② b 的值按 $p\sum_{j=1}^{50} a_j$ 计算，其中 p 为区间[0.25，0.75]内一随机数。

③ 若所有 $a_j \leqslant b(j=1,\cdots,50)$ 则结束操作；若存在 $a_j > b$，则返回到操作步骤(1)，(2)。

表 7.15 列出了按照以上步骤产生的一个 50 变量的背包问题数据。设定遗传算法运行时个体数为 50，交叉率为0.8，变异率为0.001，终止世代数为500。对每个背包问题，将遗传算

表 7.15　50 个变量的背包问题数据表

c_j	882	16	216	660	354	233	124	832	12	920
	61	273	763	406	985	562	123	972	879	603
	882	192	137	609	260	292	732	723	383	561
	620	760	848	707	117	453	640	703	599	163
	822	540	898	311	421	497	86	326	447	
a_j	600	428	590	34	241	922	794	828	853	612
	747	779	384	985	715	298	423	256	762	96
	402	842	872	446	760	122	895	163	98	855
	792	354	92	628	327	930	9	734	305	739
	774	907	508	488	839	865	486	769	368	937
b	188 337									

法模拟计算末世代目标函数值与分枝定界法获得的精确值比较得到相对误差。对 10 个不同的问题分别计算，得到表 7.16 所示的比较结果。显然，对于同一问题，二重结构编码的遗传算法比基本遗传算法要优越得多。

表 7.16 SGA 与二重结构编码的遗传算法计算结果比较

	SGA		二重结构编码的遗传算法	
	最差值的误差	最优值的误差	最差值的误差	最优值的误差
问题 1	0.076	0.054	0.001	0.000
问题 2	0.128	0.112	0.020	0.000
问题 3	0.130	0.108	0.024	0.009
问题 4	0.140	0.087	0.006	0.044
问题 5	0.037	0.033	0.016	0.000
问题 6	0.145	0.082	0.035	0.000
问题 7	0.195	0.084	0.053	0.037
问题 8	0.082	0.079	0.011	0.000
问题 9	0.173	0.104	0.040	0.020
问题 10	0.218	0.133	0.052	0.015

需要说明的是，这里，我们讨论了相对简单的单目标背包问题，如果考虑多目标的情况，即成为多目标 0－1 规划问题，可以参照 6.4 节介绍的多目标遗传算法(MOGA)来解决。

参考文献

[1] Davis L. Job Shop Scheduling with Genetic Algorithm. In: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1985, 136～140

[2] Fox B R, McMahon M B. Genetic Operators for Sequencing Problems. In: Foundations of Genetic Algorithms, Rawlins G J E(ed). Morgan Kaufmann Publishers, 1991, 284～300

[3] Grefenstettee J J, Gopal R. Genetic Algorithms for the Salesman Problem. In: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1985, 160～168

[4] Grefenstettee J J. Incorporating Problem Specific Knowledge into Genetic Algorithms. In: Genetic Algorithms and Simulated Annealing, L.Davis(ed.), Morgan Kaufmann Publishers, 1987, 42～60

[5] Nakano R. Conventional Genetic Algorithm for Job Shop Problems. In: Proceedings of the Fourth International Conference on Genetic Algorithmss, 1991, 474～479

[6] Oliver L M, Smith D J, Holland J R C. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, 224～230

[7] Whiley L D, Starkweather T, Fuquay D A. Scheduling Problems and Traveling Salesman: The Genetic Edge Combination Operator. In: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufamann Publishers, 1989, 133～140

[8] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer－Verlag, Second, Extended Edition, 1994

[9] Whitley D, Yoo N W. Modeling Simple Genetic Algorithm for Permutation Problems. In: Foundations of

Genetic Algorithms, Morgan Kaufmann Publishers, 1994, 115～137

[10] Ahramson D, J. Abela. A Parallel Genetic Algorithm for Solving the School Timetable Problem. 15th Australian Computer Science Conference, Hobart, 1992

[11] Morikawa K, Furuhashi T. Cooperation and Evolution of Scheduling System with Genetic Algorithms. Research Report, Department of Information Electronics, Nagoya University, 1996

[12] Tsujimura Y, Gen M. Genetic Algorithms for Solving Multi－processor Scheduling Problems. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 106～115

[13] 坂和, 加藤. 遺伝的アルゴリブムによるフレキブルスケジエリーング. 日本フアヅイ学会志, 1995, 7(5): 177～185

[14] Ansari N, Hou E. 用于最优化的计算智能. 李军, 边肇祺 译. 北京: 清华大学出版社, 1999

[15] 玄光男, 程润伟. 遗传算法与工程设计. 北京: 科学出版社, 2000

[16] 王黎, 马光文. 基于遗传算法的水电站优化调度新方法. 系统工程理论与实践, 1997(7): 65～69

[17] 刘民, 吴澄, 蒋新松. 用遗传算法解决并行多机调度问题. 系统工程理论与实践, 1998(1): 14～17

[18] 蔡延光. 多重运输问题的遗传算法及其遗传局部搜索. 系统工程理论与实践, 1997(12): 101～106

[19] 赵赫, 杜端甫. 遗传算法求解旅行推销员问题时算子的设计与选择. 系统工程理论与实践, 1998(2): 62～65

[20] 何翠红. 基于遗传算法的分组交换网络路由选择新算法. 系统工程与电子技术, 1998(2): 48～52

[21] 李大卫, 王莉, 王梦龙. 基于遗传算法的钢管生产批量计划. 系统工程理论与应用, 1998, 7(3): 22～26

[22] 王莉, 王梦光. 基于遗传算法的多机多阶段的 FlowShop 问题. 信息与控制, 1997, 26(4): 296～300

[23] 纪树新, 钱积新, 孙优贤. 车间作业调度遗传算法中的编码研究. 信息与控制, 1997, 26(5): 393～400

[24] 方剑, 习裕庚. 基于遗传算法的滚动调度策略. 控制理论与应用, 1997, 14(4): 589～594

[25] 李道波, 胡庆生, 林争辉. 一种适于优化分配问题的对称遗传算法. 计算机辅助设计与图形学报, 1998, 10(1): 22～28

[26] 刘德全, 滕弘飞. 矩形件排样的遗传算法求解. 小型微型计算机系统, 1998, 19(12): 20～25

[27] 唐飞, 滕弘飞. 一种改进的遗传算法在布局优化中的应用. 软件学报, 1999, 10(10): 1 096～1 102

第8章 遗传算法与机器学习

机器学习(machine learning)以复杂系统为对象,研究机器如何获取新知识或改善已有知识的问题,属于人工智能研究中较年轻的分支。由于20世纪40年代计算机的产生,使机器学习的实现有了可能,并且在50年代中期到60年代中期成了机器学习的热烈时期,这一时期主要研究各类自适应系统,并已开始研究神经网络模型和人工进化系统。60年代中期到70年代中期转入低潮,主要研究侧重于基于概念的学习和基于归纳的学习;70年代中期到80年代中期又得到了迅速地发展,特别是专家系统的成功应用,不同的学习策略和各种学习方法相继问世,示例归约学习成为研究主流,如出现了影响很大的示例学习方法:ID系列和AQ系列。此外,自动知识获取成为机器学习的应用研究目标,遗传算法应用于机器学习的思想已经被提出。最近10多年机器学习的研究和发展又进入了一个崭新时期。1986年,神经网络重新兴起,基于连接机制的学习开始向传统的符号学习(symbolic empirical learning)挑战。神经网络将知识的表达缊涵于网络连接中,处理隐层和反向传播算法的发展,显示出很强的学习能力,因而,广泛应用于模式识别、自动控制、信号处理、语音识别等许多领域。随着各种改进型学习算法不断地被提出,显著地改善了机器学习系统的性能。而此前的符号学习试图将知识由一个知识库来包含和解释,其学习能力非常局限,只能适用于知识表达相对简单的系统。

迄今,自然界只有人类才真正具有完善的学习能力,机器学习系统实际上是对人的学习机制的一种抽象和模拟,是一种理想的学习模型。基于符号学习的机器学习系统如监督型学习系统、条件反射型学习系统、类比式学习系统、推理学习系统等,只具备一些较初级的学习能力。近年来,由于遗传算法的发展,基于进化机制的遗传学习成为一种新机器学习方法,它将知识表达为另一种符号形式——遗传基因型,通过模拟生物的进化过程,实现专门领域知识的合理增长型学习,所以有的学者将之称为次符号学习方法(subsymbolic learning)。机器学习成为遗传算法的经典应用领域之一,归功于密歇根大学Holland等早期的工作。他们将基于遗传的机器学习(Genetic-based machine learning,GBML)方法发展成为CS1的分类系统(classifier system)学习方法,奠定了遗传算法重要思想的基础,后来被归纳为"密歇根方法"(Michigan approach)。1991年,De Jong等提出了"匹茨堡方法"(pitt approach)。1994年,日本名古屋大学的市桥等结合两种方法的优点提出了"名古屋方法"(nagoya approach),这些方法都分别在复杂机器学习系统中获得了成功的应用。此外,遗传程序设计方法应用于机器发现(machine learning from discovery)系统的研究以及结合不同学习方法交互作用的混合学习方法也开始受到重视。

本章将介绍基于遗传算法的机器学习一般方法,包括密歇根方法和匹茨堡方法,并分析神经学习和遗传学习的交互作用。

8.1　基于遗传算法的机器学习

遗传算法应用于机器学习领域，始于 Holland 和 Reitman 的 CS1。起初他们称之为 animat，取自 animal 和 robot 两个词的组合。其宗旨在于，使复杂环境中行走的机器人(robot)像动物一样具有高度学习能力，而这种学习能力主要地依靠遗传算法方法获得。这类对象系统要求不仅能够处理已确定的状态模式，而且能够处理新出现的状态模式。它需要一个自动知识获取和提炼过程，即先进的机器学习过程，基于这一过程获得高度的自适应能力。

机器学习分为两个大方面：其一是适当的状态空间划分，即所谓的分类(category)；其二是在各分类下的得到期望行动(action)。一般的机器学习问题，需要克服以下几个困难：

① 由于数据含噪声，并且常有新状态出现，要求机器具有健壮性和自适应能力；

② 多数要求实时的行动；

③ 目标通常比较间接或者是模糊的；

④ 系统评价规则的获得很困难。

这些困难克服的有效途径之一是研究知识的发现机制。知识可用产生式系统的规则表示，这种表示虽然简单，但计算完备，便于处理。其形式如下：

$$\text{IF} < \text{condition} > \text{THEN} < \text{action} > \tag{8.1}$$

意思是指当条件满足时，即规则着火，就可能采取行动。如果规则设计恰当，当所有规则一起工作时可以获得期望的行动。传统的产生式规则系统在学习上的障碍是其规则过于复杂，即规则的条件和行动部分都允许包含语法结构。用遗传学习的方法获得新的规则系统，一般采用固定长度规则表示，通过世代的进化获得自适应能力。通常，将它称为基于遗传的机器学习(Genetic-based machine learning, GBML)。Goldberg 在工程系统控制中应用了 GBML，一个是极点平衡问题，另一个是煤气管道压缩机系统。

GBML 系统与传统的专家系统有着明显的不同。传统的专家系统需要建造一个规则知识库，代价很高而且难以实现，并且系统很难与不断变化的环境保持一致，易出现冗余、矛盾、蕴涵等现象，更重要的是专家系统中的规则和规则相应的信度是预先由程序设计者根据专家知识给出的，是固定不变的。对于 GBML 系统，以分类器系统为例，它对应一个自适应的学习系统，使用的是概率转换规则，而不是确定性规则，其规则和相应的信度是不固定的，正是需要学习的关键信息。此外，专家系统中使用的串行触发机制，而分类器系统中使用的是并行规则触发器。

遗传算法应用于机器学习，与最优化问题的应用存在着根本的不同，最优化问题强调搜索收敛到一个近似最优解，而 GBML(密歇根方法)不仅要获得代表一条规则的好的个体，而且更加强调最佳协调的规则组合。一般而言，GBML 应具备以下几方面的机能：

① 新规则生成，遵循优胜劣汰的机制；

② 学习过程中不破坏已获得的优良规则；

③ 规则数目不预先给定；

④ 相似的或者相互包含的规则，进行适当的取舍，规则集合不过分冗余。

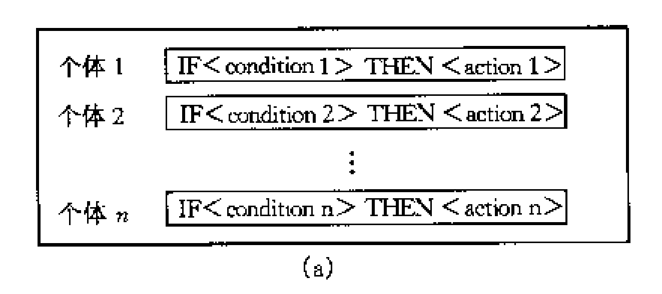
GBML 系统依靠规则间的协调，使系统发挥高性能。通常分为以下两种协调方式：

(1) **弱协调**　无论各规则单独使用，还是全部一起使用，都能确保系统的性能。规则之间

互不影响。

(2) **强协调** 只有全部规则一起使用，才能确保系统的性能。规则的删除、追加、修改对其他规则存在一定的影响。

如图 8.1 所示，GBML 的典型方法为 Holland 的密歇根方法和 De Jong 的匹茨堡方法。这两种方法根本区别在个体与种群组成上，密歇根方法将一条规则描述为一个个体，而规则集对应一个种群；匹茨堡方法视规则集为一个个体，多个规则集对应一个种群。下面两节将分别介绍这两种方法。



(a)

个体 1 IF<condition> THEN <action> … IF<condition> THEN <action> … IF<condition> THEN <action>

个体 2 IF<condition> THEN <action> … IF<condition> THEN <action> … IF<condition> THEN <action>

⋮

个体 n IF<condition> THEN <action> … IF<condition> THEN <action> … IF<condition> THEN <action>

(b)

图 8.1 密歇根方法与匹茨堡方法中的个体与种群

(a)密歇根方法； (b)匹茨堡方法

8.2 密歇根方法

1976 年，Holland 正式发表了遗传学习分类系统。此后，这方面的研究开始活跃起来，吸引了许多机器学习研究者加入其中，逐渐地成为遗传算法应用的重要领域。1994 年 Evolutionary Computation 第二卷第一期专门组织了"分类系统特辑"介绍这方面的研究进展。这里，我们主要介绍 Holland 在这方面的工作。

8.2.1 分类器学习系统的结构

Holland 的分类器系统的结构如图 8.2 所示。环境信息通过分类器系统的检测器(detector)被编码成有限长的消息(messages)，然后发往消息表(message list)；消息表中的消息触发分类器(classifier)，被触发的分类器又向消息表发消息，这些消息又有可能触发其它的分类器或引发一个行动，通过作用器(effector)作用于客观环境。

(1) **检测器** 将环境信息中由重要特征和类别组成的训练例子集，编码成二进制字符串的消息。一条消息 M^t 是一个二元组，其形式如下：

$$M^t = (x^t, y^t) \tag{8.2}$$

其中，i 为消息号；x 为条件部分，即训练例子的各特征编码，$x^i \in \{0,1\}$；y 为结论部分，即训练例子的类别，$y^i \in \{0,1\}$。例如，((10001011)，(1011))是由8位条件和4位结论组成的消息。

(2) **消息表(ML)**　包含当前所有的消息(训练子集)。

(3) **分类器(CL)**　分类器系统与传统的机器学习系统不同，它最后获得的规则中包含通配符#，这就会出现大量的冗余规则，如：1# #0与1110是一致的。一般来说，能使系统使用最小的规则集获得较高的性能。规则集越小，系统的实时性能越好。

分类器是由当前进化产生的规则组成的，一个规则 C^i 是一个三元组，形式如下：

$$C^i = (U^i, V^i, fitness^i) \tag{8.3}$$

其中，U^i 是条件部分，$U^i \in \{0,1,\#\}$；V^i 是结论部分，$V^i \in \{0,1\}$；$fitness^i$ 是规则 i 的适应值，它又是一个二元组，其形式如下：

$$fitness^i = (fit1, fit2) \tag{8.4}$$

其中，$fit1$，$fit2$ 均为正整数，分别表示在该规则覆盖的范围内，与规则结论一致和不一致的消息个数。

(4) **测试表(test list, TL)**　测试表由所有测试例子组成。一个测试例子 T_i 也是一个同消息一样的二元组，只是它的结论部分 $y^i \in \{*\}$，* 表示未确定。当执行完分类器规则后，其结论部分 y^i 就被赋值成与消息 M^i 完全一样的形式，即 $y^i \in \{0,1\}$，变成一条新的消息。结论直接作用于环境，也可通过环境将新消息反馈给系统，以便系统能够继续学习下去，从而更好地适应不断变化的环境。

(5) **作用器**　将测试的判别结果转换成具体问题的真实的输出值，并作用于环境。

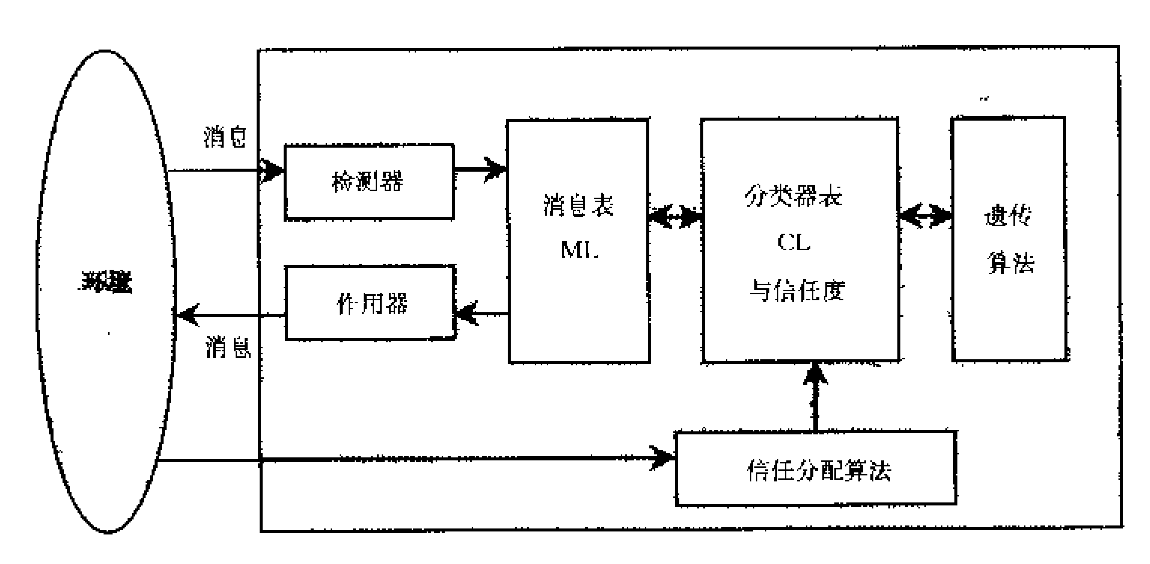


图8 2　分类系统的结构

8.2.2　分类器学习系统的主要算法

1. 信任分配算法(Credit Assignment Algorithm, CAA)

所谓信任分配是根据各分类器所起的作用对所有分类器进行排列。一般来说，分类器起的作用越大，其权值也就越大。在参与消息匹配的所有分类器中根据分类器的权值来选择响应消息的分类器。这种策略就可保证较优规则的生存和低劣规则的消亡。

进行信任分配的常用算法是桶队列算法(Bucket Brigade Algorithm)。可以将桶队列形象地看成一个消息拍卖市场,在这里分类器可以进行交易权的买卖。获得交易权的分类器就可进行消息的交易,这样在消息产生(环境)和消息的使用(作用器)之间就形成了一个由分类器作为中介的消息分配链。

在桶队列算法中采用了拍卖机制(auction)和交易(clearing)机制。在拍卖机制中,所有被匹配的分类器需根据其权值(实力)参与投标(bid),投标值与该分类器的权值成正比,投标值越高,其适应性越强,就越有可能参与信息的发送。一旦一个分类器被选中(中标)发送其消息,它必须通过交易所(clearing house)支付其投标值给消息提供者。当某条消息被一个或多个分类器所匹配,这些分类器就组成一个桶。

为了更好地说明桶队的工作,可参看表 8.1 给出的例子。在表中,$t=0$ 时,环境消息 0111 出现在分类器表中,此消息与分类器 1 匹配,并发送出分类器 1 的消息 0000。在 $t=1$ 时,消息 0000 依次与分类器 2 和分类器 4 相匹配,被匹配的分类器 2 和 4 分别发送出消息 1100 和 0001。在 $t=2$ 时,消息 1100 与分类器 3 和分类器 4 相匹配。$t=3$ 时,分类器 3 发出消息 1000,并和分类器 4 相匹配。$t=4$ 时,分类器发送出消息 0001,过程结束。

表 8.1 一个手工计算的桶队列工作过程($C_{bid}=0.1, C_{tax}=0$)

	分类器	$t=0$				$t=1$			
		权值	消息	匹配	投标	权值	消息	匹配	投标
1	01##:0000	200		1	20	180	0000		
2	00#0:1100	200				200		1	20
3	11##:1000	200				200			
4	##00:0001	200				200		1	20
	环境	0	0111			20			

	分类器	$t=2$				$t=3$			
		权值	消息	匹配	投标	权值	消息	匹配	投标
1	01##:0000	220				180			
2	00#0:1100	180	1100			218			
3	11##:1000	200		2	20	180	1000		
4	##00:0001	180	0001	2	18	162	0001	3	16
	环境	20				20			

	分类器	$t=4$			$t=5$		投标结果
		权值	消息	匹配	投标	权值	
1	01##:0000	220				220	
2	00#0:1100	208				208	
3	11##:1000	196				196	
4	##00:0001	156	0001			206	50
	环境	20				20	

下面将对分类器的权值及其随时间的变化情况进行分析。

任选一分类器 i,在时间$(t+1)$的权值 S 由下式计算:

$$S_i(t+1) = S_i(t) - P_i(t) + R_i(t) \tag{8.5}$$

式中,$S_i(t+1)$为第 i 个分类器在时间$t+1$ 时刻的权值;

$S_i(t)$为第 i 个分类器在时间t 时刻的权值;

$P_i(t)$为第 i 个分类器在中标时所有的支出;

$R_i(t)$为第 i 个分类器的回报。

由(8.5)式可知,$S_i(t+1)$取决于:分类器在时刻 t 是否中标,分类器由于以前发送的消息而得到的回报以及分类器在时刻 t 的权值。由于分类器 i 的投标值正比于其权值,因此,投标值按下式计算:

$$B_i = C_{\text{bid}} S_i \tag{8.6}$$

式中,B_i 为分类器的投标值,C_{bid}为分类器 i 的中标系数。

设 k 为消息队列的容量,即最多同时接受 k 条消息,在投标选取 k 个最好的分类器时引入一个随机扰动,即:

$$EB_i = B_i + N(\sigma_{\text{bid}}) \tag{8.7}$$

式中,$N(\sigma_{\text{bid}})$为均值为 0 且方差为 bid 的正态分布随机噪声。

为了防止某些分类器权值虽然不高,但由于随机因素却能中标,需引入征收与其权值成正比税收的策略,即:

$$T_i = C_{\text{tax}} S_i \tag{8.8}$$

因此,对任一分类器的权值计算有以下的方程:

$$S(t+1) = S(t) - C_{\text{bid}} S(t) - C_{\text{tax}} S(t) + R(t) \tag{8.9}$$

即

$$S(t+1) = (1-k)S(t) + R(t) \tag{8.10}$$

式中,$k = C_{\text{bid}} + C_{\text{tax}}$。

设 $t=0$ 时,初始权值为 $S(0)$。则在 $t=n$ 时,有:

$$S(n) = (1-k)^t S(0) + \sum_{j=0}^{n-1} R(j)(1-k)^{n-j-1} \tag{8.11}$$

当 $R(t)=R_{ss}$时,S 值趋于稳定,这时 $S(n)=(1-k)^n S(0)$。令 $S(t+1)=S(t)=S_{ss}$,则有:

$$S_{ss} = \frac{R_{ss}}{k} \tag{8.12}$$

因此,可得:

$$B_{ss} = C_{\text{bid}} S_{ss} = \frac{C_{\text{bid}} R_{ss}}{k} = \frac{C_{\text{bid}} R_{ss}}{C_{\text{bid}} + C_{\text{tax}}} \tag{8.13}$$

由于 C_{tax}远小于 C_{bid},故 $B_{ss} \approx R_{ss}$,即当分类器的回报趋于稳定时,投标值接近于回报值。

在分类器系统中,规则的权值是否处于稳定状态,对遗传算法的学习过程有一定的影响。

假设分类器 $c \in CL$,其权值为 $p(c,k)$,k 表示学习次数(时间参数)。实际使用的信任分配算法描述如下(算法示意图如图 8.3 所示,图中标注的序号对应以下的步骤号):

第 1 步 检测器从环境信息中获取训练例子集,编码成二进制字符串的消息。多个消息组成消息表 ML。

第 2 步 逐一处理分类器表 CL 中每个分类器,检查 ML 中是否存在与该分类器的条件

部分匹配的消息，如果存在匹配，将该分类器置于集合 M。

第 3 步　已有匹配的分类器 c 的投标值用下式计算：

$$B(c,k)=\frac{R(c)}{b}p(c,k) \tag{8.14}$$

式中，b 表示条件部分的位长，$R(c)$为条件部分非#码的位数。

M 中按各分类器的投标值大小决定中标者，即胜者。胜者因中标支出，其权值减少，即：

$$p(c,k)=p(c,k-1)-B(c,k)$$

这里，在前次学习中向分类器 c 发送消息的分类器 s 的权值按下式计算：

$$p(s,k)=p(s,k-1)+B(c,k)/n \tag{8.15}$$

式中，n 为向分类器 c 发送消息的分类器数目。

第 4 步　交易。

第 5 步　计算测试表 TL 中各分类器支出的投标值，将投标值在匹配的分类器中平均分摊。激活测试表中的分类器，生成新的结论，并作为新的消息置入 ML 中。

第 6 步　根据测试表 TL 得到的结论通过作用器施于环境。

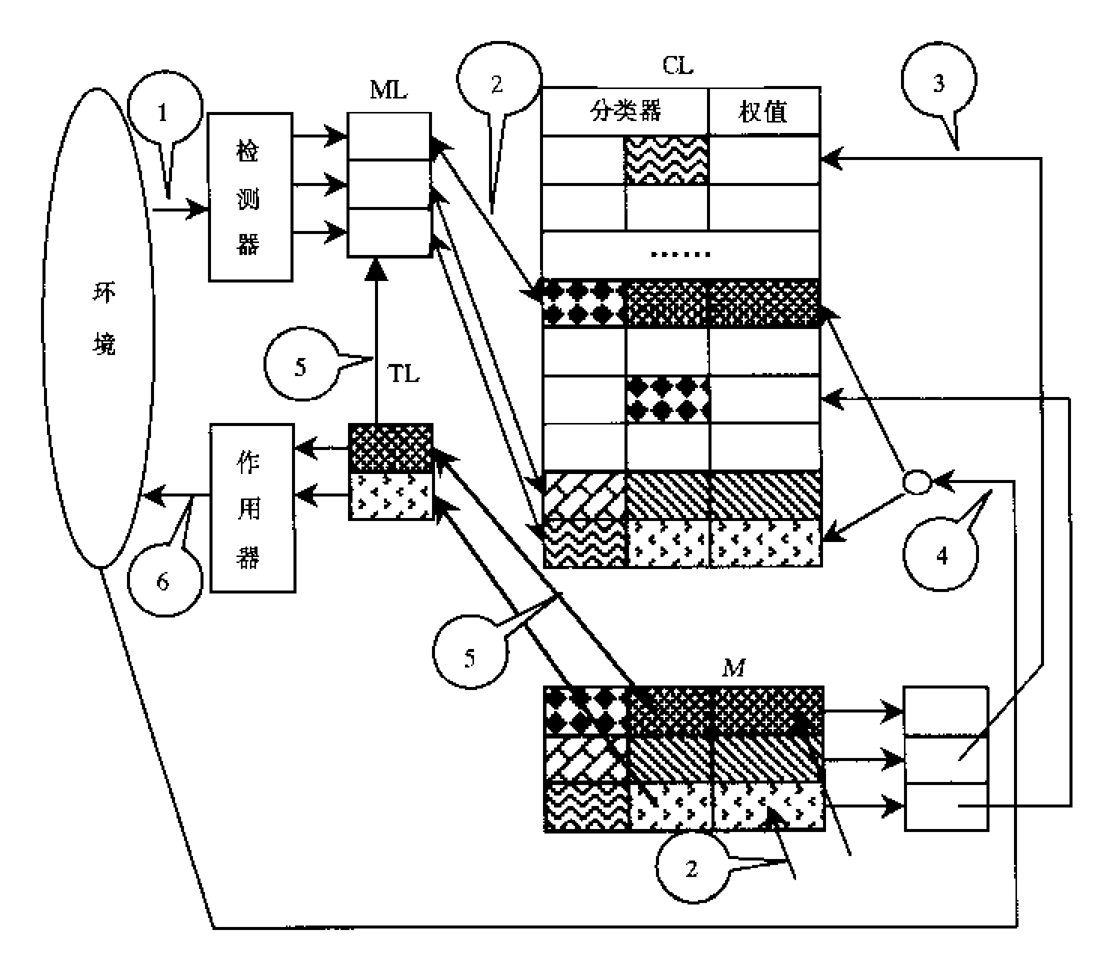


图 8.3　信任分析算法示意图

2．机器学习中的遗传算法

机器学习中的遗传算法与解决最优化问题的遗传算法不同，这里，遗传算法是用来产生新的规则。因此，利用一种限制交配策略，只允许同类(规则的结论部分相同)的规则进行交叉，

称之为本地算子。该算子对同一结论的规则，只允许其条件部分进化。如果规则的条件和结论同时进化，就可能引起种群不收敛的情况产生。变异操作一般按如下方式进行：

$$0 \to \{1, \#\}$$
$$1 \to \{0, \#\}$$
$$\# \to \{0, 1\}$$

此外，在种群进化过程中，不是对整个种群进行选择操作，一般选出 20％的个体进行遗传操作。学习过程中使用遗传算法操作约定一定的周期，例如，机器每学习 5 000 次利用遗传算法进行规则发现。

这里，我们给出一个世代的遗传算法操作步骤：

第 1 步　将 M 中各分类器作为遗传算法中的个体，计算分类器的权值，作为个体的适应度评价值；

第 2 步　轮盘赌选择；

第 3 步　实行本地交叉；

第 4 步　变异操作。

参见附录Ⅱ.2 基本遗传学习分类系统的源程序。

8.2.3　CS1

CS1 为密歇根大学 Holland 和 Reitman 设计并由 Wright 和 Forman 用 FORTRAN Ⅵ编程实现的第一个遗传学习分类系统，该分类器系统主要包括规则及消息系统、信任分配系统和遗传算法等。当时所用的计算机为 IBM1800，内存很小，只有 32 KB。设计的分类器由 25 位二进制编码构成，其中 8 位为外部输入条件，16 位为内部条件和 1 位输出。由于内存所限，只能处理 100 个分类器的分类器表，将输出为 0 和输出为 1 的分类 1 的分类器分成 50 个一组，进行信任分配。并且用遗传算法进行规则(分类器)发现。

20 世纪 90 年代，Holland 在圣菲研究所(Santa Fe Institute，SFI)工作期间，用分类系统模拟了一些具有一定适应能力的经济 Agent。经过演化，这些 Agent 都发展到能够根据一个简单商品市场的动向采取行动的程度。迄今为止，对分类系统的研究工作表明，它完全有能力处理非常复杂的行为。

值得一提的是，1996 年，西班牙格兰那达大学的 O. Cordon 和 F. Herrera 等根据密歇根方法的优点提出了一种规则迭代学习方法(iterative rule learning approach)，采用了“在重复中学习”(learning by trials)的学习策略。它具有记忆系统和经验修正机制。与密歇根方法相同，该方法将单个规则视为一个个体。但每次只有最佳个体作为问题的偏解。为了获得一个控制系统性能的规则集，采用下面的迭代学习步骤：

第 1 步　用遗传算法获取一条最佳规则；

第 2 步　将这条规则置入规则集；

第 3 步　对这条规则进行惩罚处理，置入遗传算法的种群中；

第 4 步　当前规则集经过训练样本集的检验，如果达到预期最好的性能，结束学习过程，当前规则集即为问题的解。否则，返回到第 1 步。

这种方法与密歇根方法的主要区别在于，每个个体的适应度都是独立计算的。

8.3 匹茨堡方法

匹茨堡方法将体现系统性能的全体规则集作为遗传算法的一个个体，多个规则集对应于一个种群，规则集评价对应于个体的评价。因此，可以直接应用遗传算法，将个体染色体中每条规则编码对应一个遗传基因，以规则为单位实行遗传操作的交叉和变异。这种方法存在两个困难问题，其一是以规则为单位实行遗传操作，规则数少时容易在初期收敛；规则数多时，由于基因编码过长，存在重复的规则，算法效率很低；其二是规则个数难以预先给定，基本遗传算法难以适用。围绕这两个问题，人们提出了多种改进方法。例如，对于第一个问题，采用对各规则的遗传基因内部实行可能的交叉操作，以保持遗传多样性，从而避免过早收敛。对于第二个问题，采用附加的冗长规则编码，使个体的染色体编码长度相同；或者通过改进遗传操作算子，来处理长度可变、基因座独立的基因组（染色体）。

值得一提的是，1994年，日本名古屋大学的市桥等通过改进匹茨堡方法提出了所谓的"名古屋方法"(nagoya approach)。该方法采用一种变长度染色体描述个体，如在4.3节介绍的messy GA。并借用微生物进化机制对遗传操作进行改进，依据细菌遗传学的遗传因子重组，细菌通过交配传递给接受细胞DNA物质，相互交配的遗传基因能够从一个细菌传递到其他细菌，引起快速的进化。这里，一个细菌对应匹茨堡方法的一个个体，个体的基因组由若干个基因片段组成，每个基因片段对应一条规则。首先，一个细菌被克隆为m个，对基因组中相同的基因片段实行变异。其方法是先选择一个细菌中最佳基因片段，并将它传递到其他$m-1$个细菌，取代它们中最差的基因片段，然后基因组中其他基因片段实行变异。基因片段变异之后，再对细菌种群执行选择操作和交叉操作，采用messy GA的方法。这种方法实际上结合了一个新的局部改善机制，比较有效地解决了匹茨堡方法的两个困难。该方法成功地应用于模糊控制器规则学习的系统中，示例详见9.4节。

8.4 学习与进化之间的交互

对一般系统而言，系统的环境或系统的内部结构发生变化后，如果能够满足性能函数的要求，那么我们就称这个系统是适应性的；如果系统的环境发生变化，且经过时间T后，能够满足性能函数的要求，那么我们就称这个系统为学习系统。其实，这两个概念的认识最早来源于生物界。现在，越来越多的人已认识到，借鉴生物的适应和学习机制，有助于建构具备自学习、自适应能力的人工智能系统。将遗传学习(genetic learning)与生命周期学习(life－time learning)结合起来研究遗传机器学习系统和人工生命系统，已成为目前非常活跃的研究课题之一。其主要原因有以下三个方面：

① 适应性行为与学习密不可分，生物学习与进化之间交互作用贯穿于生物适应性行为获得的漫长历史中。将基因延续的遗传学习和生命周期学习相结合，更加贴近于自然。

② 将遗传搜索与局部搜索相结合，可以有效地改善遗传算法的搜索性能。生命周期学习便是一种局部改善机制。

③ 常规遗传算法中适应度的评价与生物学中适应度的概念是有一定差异的。常规的遗

传算法中个体的适应度计算是基于个体适应状态的评价，且在个体产生之后随即进行评价，忽略了个体在整个生命期内需要承受连续不断的适应性考验。而这些考验不是预定义好的程序，它们来自于自然，不仅受自身行为的影响，而且受到其他个体的影响，局部或整体环境的变化也会影响到个体的适应性。

从生物进化的角度讨论生物适应性与学习关系的一般性问题，研究遗传学习与生命周期学习相结合的方法，对设计自适应系统具有特别的意义。1995年，荷兰学者Jan Paredis提出了协同进化计算的思想，他在这方面的探索研究为我们提供了新的启迪。

8.4.1 学习与进化的交互作用

学习和进化可以视为两类不同的生物适应过程，前者是发生于生物组织器官生命周期内，后者产生于生命的进化历史长河。问题在于，如何衡量个体的适应性？个体生命周期内的学习历程是否对物种的进化起引导作用？如果存在这样的作用，其相互影响方式又如何？

迄今为止，衡量生物适应性有以下三种不同的标准：

(1) **“生存”标准** 达尔文用“适者生存”来表述自然选择的结果，“生存”是适应的标准。

(2) **“繁殖”标准** 现代综合进化论认为“生存”不能用来作为定量地衡量适应的程度，因为生存和死亡是“全或无”的概念。应以“繁殖”或基因延续作为标准。

(3) **生态学标准** 生态学家认为“繁殖率”和“存活率”是生态学的指标，不能作为适应的衡量指标，从生态学角度看，生物对环境资源的利用效率是一客观的标准。但这种标准只有相对意义。

从大进化的角度，上述三个标准都不适用。现在，人们越来越趋向这样的认同，适应应该理解为生物的某种状态(即结构和功能特征符合生物生存或延续的利益)以及生物获得这种状态的过程。事实上，生物的不同适应特征(状态)有不同的进化历史。

一般认为，生物学习方式是系统发育的学习和个体发育的学习的综合，实际上是基因延续的遗传学习和生命周期学习的结合。1961年，N. Wiener将具有学习能力的系统定义为“在其生命周期间能够根据过去的经历和环境来改造自身的系统”，他指出：“一个动物进行学习，那么它就是一个能够被它过去的环境转变为另一个不同动物的动物。因而，在它生活期内对环境的影响是可以调节的。一个动物进行繁殖，就是它能够产生出另一些虽然不是与它完全相同，至少是近似相同的动物。所谓近似相同，指的是在时间的进程中可能发生某些改变。如果这种改变本身是可以遗传的，我们就有了供自然选择发生作用的原始材料。如果行为方式具有遗传不变性，那么，在各种有了变异的行为类型中，那些对于种族的继续生存具有好处的行为方式就会传播开来，另外一些不利于种族生存的行为方式则会被消灭。这个结果就是种族的或系统发育的学习，它是与个体学习相对应的。个体发育的学习和系统发育的学习都是动物根据周围环境调节自己的方式。”

我们知道，一个人后天的学习不会影响到他的遗传基因，因此习得也不会直接传递到其后代。但进化生物学家认为学习对进化的存在间接效应，称之为“鲍德威效应”(Baldwin effect)。如果学习能够帮助生存，那么具有最善于学习的组织器官的个体会产生最适应的个体，这就增加了伺于学习的基因得以延续下来的可能。如果赖以生存的环境稳定，习得也可保持稳定，将会间接地产生这样的遗传基因，使他掌握因生存必须学习而且能够学会的东西。实际上，没有这样的学习，生存的机会就会减少。因此，学习以某种间接的方式影响进化。在模拟生物捕食行为(详见10.3节内容)我们不难解释这种“鲍德威效应”，模拟生物要维持生存必须在虚拟环

境中学会接近食物，生物个体具备这样的学习能力愈大，其能量愈大。学习的规则有两条："接近食物的个体是好的"、"如果距离食物近就向食物方向移动"，虽然我们没有将这样的学习规则直接纳入个体遗传编码或者生存评价中，但经过若干代的进化和学习，获得的最佳个体表现出很强的学习能力。

神经网络与遗传算法相结合的研究是学习和进化之间交互作用的一种模拟。图 8.4 表示神经网络的遗传设计过程，神经网络算法负责学习建模，而遗传算法构成进化建模，是神经网络学习能力的进化模拟。遗传算法的一个个体代表具有某种学习能力的一个神经网络，两者通过基因型和表现型表达，并用编码方法转换。随机产生的一组神经网络构成初始种群，种群的个体在进化的每一代都接受某种学习，经过遗传操作生成新一代个体，学习能力强的较优个体有更多的子代个体。如此学习和进化若干代，可以获得最佳的个体。

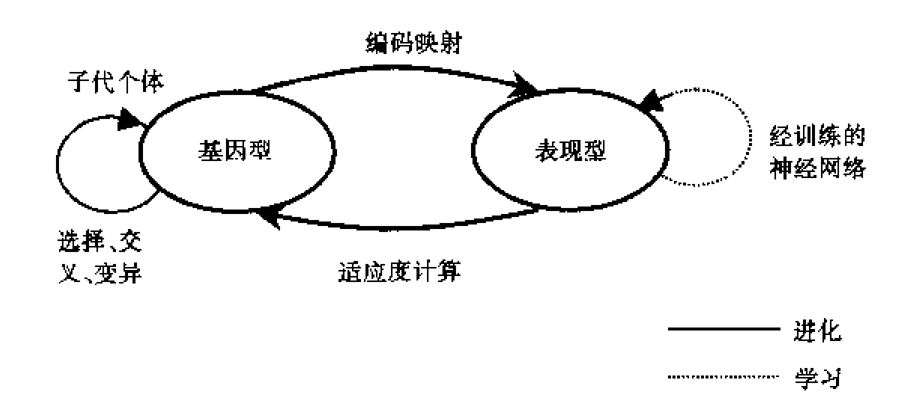


图 8.4　神经网络的遗传设计过程

在整个设计过程中，将学习和进化视为两个级别的适应过程，避免仅采用反向传播学习法设计神经网络可能产生的"过学习"现象。

每个生物个体可以作为一个 agent。agent 由两个相互协作的前馈神经网络表示，一个称为"评价网络"，将 agent 在时间 t 上的状态映射成表征其状态优劣程度的数值，实际上是"趋食性"评价，用来学习第一条规则"接近食物的个体是好的"。另一个称为"行动网络"，用来将 agent 在时间 t 上的状态映射为采取的相应行动，实际上用来模拟"趋食"行为，即学习第二条规则"如果距离食物近就向食物方向移动"，这里的行动只指为个体移动行为，当然移动可能直接导致个体间的攻击、交配，或者获取食物的行为。所有 agent 的神经网络构造相同，但连接权值存在差异。进化之初的"评价网络"代表 agent 的先天能力，经过若干代后连接权值会朝向"好"的方向发展；而在 agent 的生命周期内"行动网络"的权值可以采用增强学习算法(Reinforcement Learning Algorithm, RLA)来学习，agent 的"趋食性"评价可以作为"行动网络"权重学习的增强信息。

agent 的染色体由"评价网络"的连接权值和"行动网络"的初始权值编码构成。进行了三组试验：第一组试验同时选择种群的进化和 agent 个体的学习；第二组试验只选择种群的进化，不选择 agent 个体的学习；第三组试验只选择 agent 个体的学习，而没有种群的进化过程。从生物的平均灭绝时间结果统计，第一组的平均灭绝时间很长，第二组次之，第三组则远远次之。在第一组试验中，我们还发现，agent 在进化初期"评价网络"权重的基因码变化相对不大，

而"行动网络"初始权值的基因码变化较大,这说明了学习"趋食性"目标的重要,实际上体现了学习对生存的重要。进化后期"行动网络"初始权重的基因码变化相对稳定,这说明了在进化后期遗传的主导作用超过了学习,"行动网络"包含的行为知识是在先天遗传编码信息基础上经过若干代后天学习提炼而成的。

8.4.2　协同进化遗传算法

自然界生物之间维系着自然选择的各种反馈机制,这些反馈机制构成了生物界复杂性的驱动力。其中食物链关系(predator - prey relations)是最为重要的一种反馈,对于被捕食者而言,它们通过各种方式提高防御能力,例如,在漫长的进化过程中,鹿奔跑得越来越快,刺猬长起了带刺的盔甲,孔雀披上了美丽的羽毛,许多弱小的动物为躲避强者袭击"染"上了保护色;而对于捕食者而言,它们则靠发展自己的攻击力作为生存的前提,例如有越来越犀利的爪牙、越来越敏锐的感觉,很多动物能集群猎取食物,如狼、狮、蚂蚁等。即便如此,在这样的"军备竞赛"中,捕食者和被捕食者处于互为消长的平衡状态,任何一方的进化改变都有一个限制,即不致于造成对方的延续中断以致双方都不能生存。此外,生物界的共生(symbiosis)关系也相当普遍。例如,有很多动物在生活领域中占领一块土地或空间作为个体或集群生活繁殖的领地(territory),一旦有其他动物或本种但不属于本群的动物入侵,领地主人就要以各种方式把入侵者驱赶出去。大量证据表明,这种领地行为收益多、损失少,能够提高生物适应能力和赖以进化的繁殖能力。

关系密切的生物,如花和采粉的动物、寄生虫和寄主、捕食者和被捕食者等,一方成为另一方的选择力量,因而在进化上发展了相互适应的特性。当被捕食者发生变异、防御能力提高时,捕食者也相应地要产生对付防御能力的机制,否则就要因不能适应新的条件而被淘汰。这种以食物链关系、共生关系等为基础的生物进化现象称为协同进化(coevolution)。协同进化行为在生物界是一种普遍的现象,同时也是一种进化动力学中含增益的反馈关系。有学者形象地评价协同进化是"生态的舞台,进化的表演"(the ecological theater and evolutionary play)。物种是演员,物种进化是演员在一定的舞台(生态环境)上进行的表演,这种进化表演受到舞台背景的制约,舞台背景也要与演出内容相协调。

1992年Hillis最早将一种寄生虫和寄主协同进化的机制应用到优化搜索的改进中,提出了协同进化遗传算法(Co-evolutionary Genetic Algorithm, CGA)的思想。与常规遗传算法不同的是,CGA有两个以上协同进化的种群,对应于生物界食物链中的一环或多环。以两个种群为例,两个种群之间存在相背的适应度评价,即一方的成功正好成为另一方的失败,这样的评价机制会导致两个种群相互竞争,种群集体行为的复杂性在进化模拟中进一步得到发展。Jan Paredis引入生命周期适应度评价(Life-time Fitness Evaluation, LTFE)的方法,使CGA健壮性更好,成为一种细粒度遗传算法,获得了一系列问题的成功应用,如分类神经网络设计问题、约束满足问题等等。下面我们结合分类神经网络训练问题讨论CGA的算法和应用。

图8.5给出了一个分类问题,目的是要设计一个神经网络学习$[-1,1]\times[-1,1]$内分区到类别$\{A,B,C,D\}$之间的映射关系,随机给定200个训练样本,如$(-0.6,0.9,D)$为一个训练样本,说明点$(-0.6,0.9)$属于类别D。假定神经网络是一个标准的三层前馈网络,输入结点两个,分别为待判别点的坐标x,y,隐结点层有15个结点,输出结点4个,分别对应4个分类A,B,C,D。神经元的传递函数采用标准的Sigmoid函数。

常规的遗传算法用于该神经网络的训练问题,通常将神经网络个体描述为所有连接权值

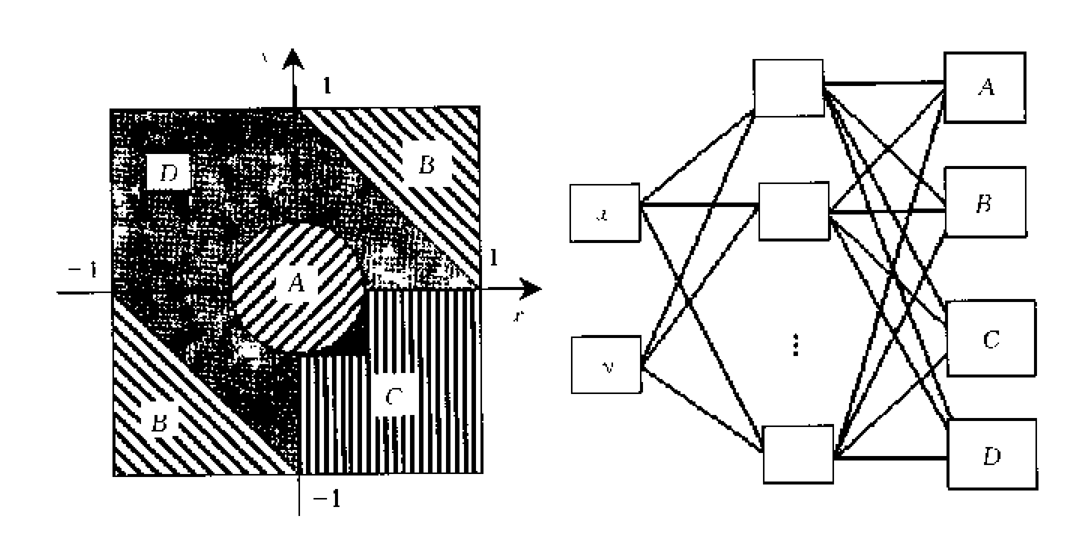


图 8.5　一个分类神经网络问题

的编码,个体的适应度可以简单定义为 200 个训练样本中正确分类的数目。首先随机产生权值的 100 个神经网络个体作为初始种群,然后进行选择、交叉和变异操作,产生新一代的个体,经过若干代的进化,获得最佳的神经网络个体。这一训练算法的实算结果表明,为获得很好的分类神经网络,需要大量的计算工作,算法的效率很低。因为大部分计算时间花在适应度评价上,每个个体适应度是在对所有 200 个训练样本测试后获得的。而大部分的测试信息是没有多大帮助的。因为进化初期大部分的训练样本会被错误地分类,进化后期大部分的训练样本又会被正确地分类,只有少部分样本是起进化导向作用。由此可见,神经网络个体测试工作的鉴别力比较低。

协同进化遗传算法用两个种群来模拟生物的"军备竞赛",种群之间的相互作用通过适应度评价来实现。对于上述神经网络的训练问题,一个种群用来包含神经网络个体,称之为解种群(solution population),解种群中的个体称为解个体;另一个种群由 200 个训练样本组成,称之为测试种群(test population),测试种群中的个体称为测试个体。我们先不考虑种群间共生关系的情况,这时,测试种群不属于一种正式的种群,测试个体在进化期间不会发生替换,只发生适应度的变化。解个体的适应度评价根据其生命周期内遭遇的测试个体的结果而定,例如,将一个解个体的适应度定义为在最近 20 次遭遇中符合解个体正确分类要求的测试个体数目。相应地,测试个体的适应度也可定义为其在最近 20 次遭遇解个体时,违反测试个体结果的解个体数目。测试个体与解个体在时间上连续成对地遭遇,每次遭遇都会改变双方的适应度大小。因此,一个测试如果在一次遭遇中分类结果正确,则其适应度反馈值为 0;反之,适应度反馈值为 1。这种适应度评价含有连续的正反馈(positive feedback)机制,因此称为生命周期适应度评价。

完整的协同进化算法可以描述为以下几个步骤:

第 1 步　产生初始的解种群和测试种群,初始解个体随机产生,测试个体根据具体问题给定。

第 2 步　初始解个体的适应度按随机挑选的 20 个测试个体中符合的个数计算,类似地,测试个体的适应度按随机挑选的 20 个解个体中违反的个数计算。实际上,对每个初始个体安排了 20 次遭遇。

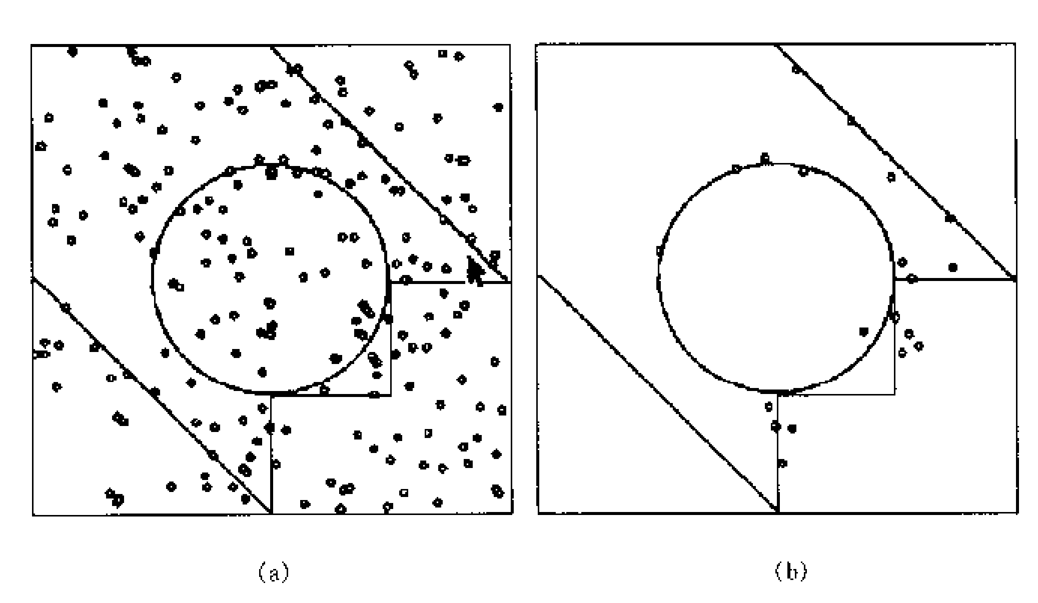
第3步 从两个种群中分别随机地挑选20个个体进行配对，并安排遭遇。在某对遭遇中，如果测试个体属于符合的情况，解个体的适应度增益为1，否则为0。同时将该解个体前20次遭遇中最早一次遭遇对象从它的遭遇历史记录中除去，然后更新一次解个体的适应度。类似地处理遭遇中的测试个体，与解个体不同的是，符合时测试个体的适应度增益为0，否则为1。

第4步 对解种群执行选择、交叉和变异的遗传操作，产生新一代的解种群。子代解个体的适应度按其与第3步中挑选的20个测试个体进行遭遇所有增益之和的计算。

第5步 若满足进化终止条件，即达到一定的世代数，或解种群中最佳解个体的适应度达到一定程度的收敛，计算结束并输出最佳解个体。否则返回到第3步，继续进行新一代的演算。

上述算法中生命周期适应度评价方法实质上体现了食物链关系的正反馈机制，这不仅减少了计算工作量，而且能够改善遗传搜索的效率。由于实际发生进化的只有解种群，这非常适合于人工智能和计算机科学中一类应用很广泛的"解－测试"问题，即在满足预定约束、准则条件下解的搜索问题，如约束满足问题、神经网络训练问题等。

对于上述分类的神经网络训练问题，分别应用常规遗传算法和协同进化遗传算法进行实算。如果将一个训练样本输入到一个前馈神经网络获得输出结果的计算过程称为一次传算。由于有200个训练样本，产生一个新个体并获得其适应度时，常规遗传算法需要200次传算，而协同进化遗传算法，只需要40次传算。将达到一定分类精度时需要完成的传算次数作为算法性能比较的依据，协同进化遗传算法比常规遗传算法要优越得多。图8.6(a)表示初始解种群200个各个体分布的情况，图8.6(b)为测试种群进化中某世代20个个体分布的情况。随着进化的发展，测试种群的个体分布于分类边界附近，为解种群的进化起导向作用。表8.2给出了两种算法性能的比较结果。从表中可以看出，CGA比GA计算工作量大大地减少，同样达到90%的分类精度，CGA的传算次数只有GA的1/3。CGA不仅收敛速度快，而且收敛分类精度也比GA高。



(a) (b)

图8.6 分类神经网络协同进化

(a)初始解种群个体分布； (b)测试种群进化中20个个体分布

表 8.2 常规遗传算法和协同进化遗传算法应用于分类神经网络训练问题传算次数对比

	>75%	>80%	>85%	>90%	>95%	平均收敛分类精度
GA	520k	920k	1 320k	2 420k	—	93%
CGA	162k	342k	502k	762k	1 902k	96.45%

参考文献

[1] Goldberg D E, Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA, Addison - Wisely, 1989

[2] De Jong K. Genetic - algorithms - based Learning, Machine Learning: An Artificial Intelligence Approach, Vol. III, Morgan Kaufmann, 1990, 611~638

[3] Grefenstette J J. Multilevel Credit Assignment in A Genetic Learning System. In: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1987, 202~209

[4] Holland J H. Genetic Algorithms and Classifier Systems: Foundations and Future Directions. In: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1987, 82~89

[5] Riolo R L. Bucket Brigade Performance I: Long Sequences of Classifiers, Genetic Algorithms and their Applications. In: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1987, 184~195

[6] Riolo R L. Bucket Brigade Performance II: Default Hierarchies, in Genetic Algorithms and their Applications. In: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1987, 196~201

[7] Wilson S W, Goldberg D E. A Critical Review of Classifier Systems. In: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1989, 244~255

[8] Wilson S W. Knowledge Growth in an Artificial Animal. In: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Publishers, 1985, 16~23

[9] Wilson S W. Classifier System Mapping of Real Vectors. In: Proceedings of 1st International Conference on Learning Classifier Systems, Houston, Texas, 1992

[10] Williams R D(ed). Two Approaches to Machine Intelligence. Computer, 1992, 78~81

[11] Collins R J. Studies in Artificial Evolution. Doctoral Dissertation, Artificial Life Laboratory, Department of Computer Science, University of California, 1992

[12] Paredis J. Coevolutionary Computation, Artificial Life 2, MIT Press, 1995, 355~375

[13] Paredis J. Coevolutionary Life - time Learning. Parallel Problem Solving from Nature IV. In: Proceedings of the International Conference of Evolutionary Computation, Voigt H M(ed). Springer Verlag, 1996

[14] Nilsson N J. Artificial Intelligence: A New Synthesis. Morgan Kaufman & 机械工业出版社, 1999

[15] 田盛丰 等. 人工智能原理及其应用. 北京: 北京理工大学出版社, 1993

[16] 史忠植. 高级人工智能. 北京: 科学出版社, 1998

[17] 史忠植. 智能主体及其应用. 北京: 科学出版社, 2000

[18] 杨炳儒 主编. 知识工程与知识发现. 北京: 冶金工业出版社, 2000

[19] 陈文伟. 智能决策技术. 北京: 电子工业出版社, 1998

[20] 孔繁胜. 知识库系统原理. 杭州：浙江大学出版社，2000
[21] 徐洁磐，马玉书，范明. 知识库系统导论. 北京：科学出版社，2000
[22] 陈阅增 主编. 普通生物学——生命科学通论. 北京：高等教育出版社，1997
[23] 张昀. 生物进化. 北京：北京大学出版社，1996 年
[24] 王彤，石纯一. 一种解释学习系统的模型 EBL/GA. 计算机学报，1997，20(2)：125～132
[25] 张雪江，朱向阳 等. 基于退化演化算法的知识获取机制的研究. 控制理论与应用，1998，15(1)：93～98
[26] 孙承意，谢克明，陈明琦. 基于思维进化机器学习的框架及新进展. 太原理工大学学报，1999，30(5)：454～457
[27] 高春华，李人厚. 基于混合遗传训练方法的分类器参数设计. 西安交通大学学报，1998，32(3)：35～36
[28] 王遵亮，吴新根，罗立民. 基于遗传算法的肝病诊断学习系统. 东南大学学报，1999，29(3)：106～109
[29] 蔡自兴，徐光佑. 人工智能及其应用(第 2 版). 北京：清华大学出版社，1996

第9章　遗传算法与智能控制

许多控制领域问题，当考虑到系统优化、自适应、自组织和自学习等方面的要求时，一般存在着许多常规方法难以奏效的困难。例如，当有非连续评价函数、缺少初始信息、模型参数和结构或控制过程中的随机性、不确定性等复杂因素出现时，现有的控制理论和方法往往因需要求导信息、对初始条件的敏感性、对高品质的领域知识依赖性而难以得到很好的应用。智能控制的概念最初是在对基于模式识别的学习控制系统的研究中逐渐形成的。1971年K. S. Fu（傅京孙）从研究自学习控制系统入手，将智能控制概括为自动控制与人工智能的交集；1977年，Saridis以智能机器人的控制为主要研究背景，从研究机器智能的角度提出了智能控制是自动控制、人工智能及运筹学的交集，并提出了分级递阶智能控制的结构和方法；Astrom提出的专家智能控制则是将专家对被控对象和控制过程的知识、经验等融入控制器的设计与控制策略中。早期的智能控制研究受到传统控制理论的影响，大部分着眼点仍然基于系统已有的先验知识来解决问题，而不是自动获取知识。模糊控制基于模糊集理论，模拟人的近似推理和决策过程。1974年Mamdani成功地将它应用于高炉和蒸汽机的控制，随后获得了迅猛发展。模糊控制的主要困难在于控制规则的获取以及控制系统的稳定性问题。20世纪80年代中期以来，神经网络得到重新重视和发展。它主要模拟人脑的形象思维以及学习和获取知识的能力。神经网络应用于控制问题，由于其高度依赖于生产现场所提供的大量训练样本，且训练算法的可实现性、实时化要求等因素，影响了其实用性。此外，对复杂问题的神经网络结构最优设计一直缺乏有效的方法。90年代以来，模糊理论、神经网络、专家系统、遗传算法和混沌方法等在许多控制领域得到了应用和发展，并且更重视这些方法相互融合，形成所谓的混合软计算（hybrid soft computing）。

遗传算法借助搜索机制的随机性，能够搜索问题域的全局最优解，并且对噪声和变化表现出很好的健壮性和自适应能力。遗传算法在控制领域的应用大致分为两类：一类是离线设计和分析；另一类是在线自适应调整。前者遗传算法作为优化器，对于已知的控制对象寻求合适的控制规则以满足控制品质方面的要求，或者对于特定的控制器结构搜索最优的参数。后者遗传算法作为一种学习机制来确定未知的或非稳定系统的特征，或者对控制器进行自适应调整。

本章首先将讨论遗传算法在系统辨识、系统控制中应用的一般方法；然后介绍进化计算与模糊逻辑、神经网络结合方法及其在智能控制中的应用，最后介绍混合软计算方面的有关研究和发展趋势。

9.1　线性系统辨识

控制工程、信号处理以及机器学习的许多问题涉及到系统辨识，其任务是从给定的输入输出数据集中确定合适的模型，作为黑盒子系统的预测和控制。线性系统辨识方法使用得最为

广泛。比较成功的方法为正交 LSR 方法(Chen 等,1989),但因搜索空间的无限增长造成的复杂性难以实际应用。Fonseca 等(1993)实现了一种将采用 GA 的子集选择和 LSR 结合起来的方法,GA 用来从可能的非线性项中选择固定数目的项,LSR 用来辨识这些项的参数。

自适应滤波的中心问题是确定模型结构及参数的最优解,以保证滤波器的性能如误差函数最小。因此,这里自适应问题可以归约为一种优化问题。线性自适应滤波器可分为自适应有限脉冲响应(FIR)滤波器和自适应无限脉冲响应(IIR)滤波器两种。物理可实现的自适应 IIR 滤波器具有更好的模型匹配和自适应能力,但其性能指标函数容易陷入局部极小点以及滤波器稳定性问题。许多学者提出了多种自适应算法,如误差算法(Output Error Algorithm, OEA)、Steiglitz－Mcbrid Method(SMM)、方程误差算法(Equation Error Algorithm, EEA)和混合梯度算法(Composite Gradient Algorithm, CGA)等,并对其收敛性进行分析,如超稳定和平均分析,但是仍不能解决自适应 IIR 滤波器的误差收敛问题。EEA 不能保证给出稳定的系统模型,OEA 存在均方输出误差的局部极小问题,SMM 算法不能处理非白加性噪声,CGA 只适合于低阶自适应 IIR 滤波,基于超稳定的平均分析理论的自适应算法虽然能解决局部极小问题,但一般不能预知其收敛条件是否满足。采用常规的梯度算法设计 IIR 滤波器,当滤波器极点处于单位圆附近时,直接实现形式的滤波器稳定性就难以保证。

近年来,人们提出了基于遗传算法、进化规划以及模拟退火等自适应滤波器设计法。利用遗传算法建模可同时确定模型结构和参数。对于线性模型,可同时获得系统的阶、时滞及参数值。其基本做法是:将相关模型结构和参数组合成相应的基因型,将拟合误差转换成相应的适应度,于是,系统建模问题就转换为利用遗传算法搜索最佳基因型结构的问题。将遗传算法用于自适应滤波不仅解决了系统的稳定性问题,而且可以实现滤波器性能函数的全局寻优和快速收敛。遗传规划方法特别适合于评价准则函数为高度非线性、存在很多局部极值点的场合,将含自适应变异的进化规划应用于滤波器的优化设计,可以保证进化滤波器的稳定性。并且可以直接利用其搜索全局最小的特点来对不同实现形式的滤波器建模。下面以 IIR 滤波器设计为例,介绍进化规划方法在系统辨识中的应用。

9.1.1 自适应 IIR 滤波器模型

IIR 滤波器可以为直接形式(direct form)、级联形式(cascade form)、并联形式(parallel form)或 Lattice 结构形式。M 阶 IIR 滤波器(M>2)的直接实现形式的传递函数为下式:

$$H(z)=\frac{a_0(n)+a_1(n)z^{-1}+\cdots+a_M(n)z^{-1}}{1+b_1(n)z^{-1}+\cdots+b_M(n)z^{-1}} \tag{9.1}$$

上式中 $a_0(n)\sim a_M(n)$和 $b_1(n)\sim b_M(n)$分别表示前馈和反馈系数。IIR 滤波器的直接实现形式模型如图 9.1 所示。由于数字系统的字长总是有限的,因此所表示的系数是有限精度的,所以直接实现形式往往有较大的累积误差。

$H(z)$等价的级联形式为:

$$H(z)=q\prod_{k=1}^{W}\frac{1-a_{1k}(n)z^{-1}-a_{2k}(n)z^{-2}}{1-b_{1k}(n)z^{-1}-b_{2k}(n)z^{-2}} \tag{9.2}$$

$H(z)$的并联形式为:

$$H(z)=p+\sum_{k=1}^{W}\frac{1-a_{1k}(n)z^{-1}-a_{2k}(n)z^{-2}}{1-b_{1k}(n)z^{-1}-b_{2k}(n)z^{-2}} \tag{9.3}$$

(9.2)和(9.3)式中当 M 为奇数时 $W=(M+1)/2$;M 为偶数时 $W=M/2$;q、p 为常数。

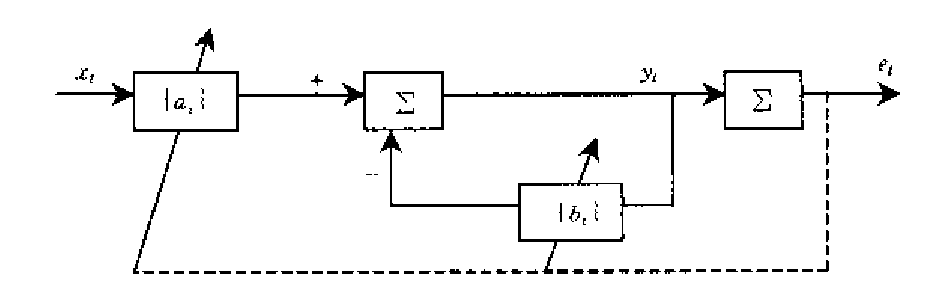


图 9.1　自适应 IIR 滤波器直接实现形式模型

当 M 为奇数时，级联形式和并联形式结构中的最后一级是单极点子系统，其余子系统或为单基点或为两极点。这两种形式滤波器的稳定性通常通过约定系数 $b_{1k}(n)$和 $b_{2k}(n)$使之处于稳定三角区来保证。

(9.1)式还可以 Lattice 结构形式实现。(9.4)式表示为 t 时刻输入输出关系，其中 $v_i(n)$ 和 $k_i(n)$为 Lattice 结构系数，当 $k_i(n)$均小于 1 时滤波器是稳定的。

$$y(n)=\sum_{i=0}^{M}v_i(n)B_i(n) \tag{9.4}$$

这里，

$$B_i(n)=B_{i-1}(n)+k_i(n)F_{i-1}(n)\qquad i=M,\cdots,1 \tag{9.5}$$

$$F_i(n)=F_{i+1}(n)-k_i(n)B_{i-1}(n-1)\qquad i=M-1,\cdots,1 \tag{9.6}$$

$$F_M(n)=x(n) \tag{9.7}$$

$$B_0(n)=F_0(n) \tag{9.8}$$

$x(n)$，$y(n)$分别为输入信号和输出信号。$B_i(n)$和 $F_i(n)$分别为时刻 n 时第 i 级 Lattice 的反馈和前馈留数(预测误差)。

上述 IIR 滤波器的不同实现形式在计算复杂性、稳定性、收敛速度以及抗干扰性等方面各有优缺点。分析不同实现形式的最小均方差(MSE)性能曲面，有助于了解有关收敛性和最小均方差的变化情况。当一个有单峰 MSE 曲面的直接形式 IIR 滤波器转变成另一种实现形式，新结构形式的 MSE 曲面具有一些附加的稳定点，它们可能是等价局部极小点或鞍点，对应于参数空间的非稳定解。级联形式和并联形式在外部传递函数与系数空间之间不存在唯一的映射关系，这种不惟一性造成了性能曲面上的附加鞍点，因而影响了基于梯度的搜索方法的效率。Lattice 结构实现形式的主要目的是使监控滤波器的稳定性更为简单化。当 Lattice 系数 $k_i(n)$均小于 1 时可以确保滤波器的稳定性，并且 Lattice 结构提供了唯一的系数映射关系，避免了分析具有不唯一映射关系的局部极小点和鞍点。

9.1.2　IIR 滤波器设计的进化规划方法

进化规划方法应用于 IIR 滤波器的优化设计问题，可以包括不同实现形式的模拟计算。具体地采用以下模拟方案：

① 进化规划引入自适应变异方法；

② 种群大小为 50，竞争数目 $q=10$；

③ 个体的变量由前馈系数 a_i 和反馈系数 b_i 构成。对于 Lattice 滤波器系数 k_i 和 v_i 作为个体的变量。

④ 进化规划引入自适应变异方法；

⑤ 对每个计算实例，种群初始化时，前馈系数 a_i 由[-1,1]区间均匀分布随机数发生器产生，反馈系数 b_i 设置为 0，以保证初始滤波器的稳定性；

⑥ 将滤波器的性能－均方差函数作为个体的适应度评价，取 100 个样本计算均方差 *MSE*：

$$MSE = 10\log_{10}\left(\frac{1}{100}\sum_{t=1}^{100}(d_t - y_t)^2\right) \quad (\text{dB}) \tag{9.9}$$

⑦ 对每个计算实例，进化模拟终止代数为 30 000 次，最终计算结果取 50 次独立模拟结果的平均值。

作为一个辨识实例，模型 I 将直接实现形式和 Lattice 结构形式应用于辨识以下未知模型：

$$H(z) = \frac{0.5 - 0.4z^{-1} + 0.89z^{-2}}{1.0 - 1.4z^{-1} + 0.98z^{-2}} \tag{9.10}$$

该模型有两个极点 p_1 和 p_2，且 $|p_1| = |p_2| = 0.9899$，说明两极点靠近单位圆。许多梯度算法对此系统的辨识存在困难。采用自适应进化规划方法可获得很好的收敛结果。如图 9.2 所示，进化过程收敛很快，直接实现形式的 *MSE* 收敛于 -75 dB，Lattice 结构形式的 *MSE* 收敛于 -114 dB，级联形式和并联形式的收敛情况与直接实现形式相同。图 9.3(a)～(c)为进化代数分别为 1，50，1 000 时最优个体的局部 *MSE* 曲面，图 9.3(d)为最优解附近全局 *MSE* 曲面，当搜索趋近于不稳定区域时，*MSE* 值急剧增大，曲面出现锯齿形状。随着进化过程的发展，大约在100代后最优解所处的局部出现，并且更多的不稳定区域也逐渐被发现，最优解附近的 *MSE* 曲面演变为涟波形状。在进化末代整个种群收敛于一个全局最小，对应于图 9.3

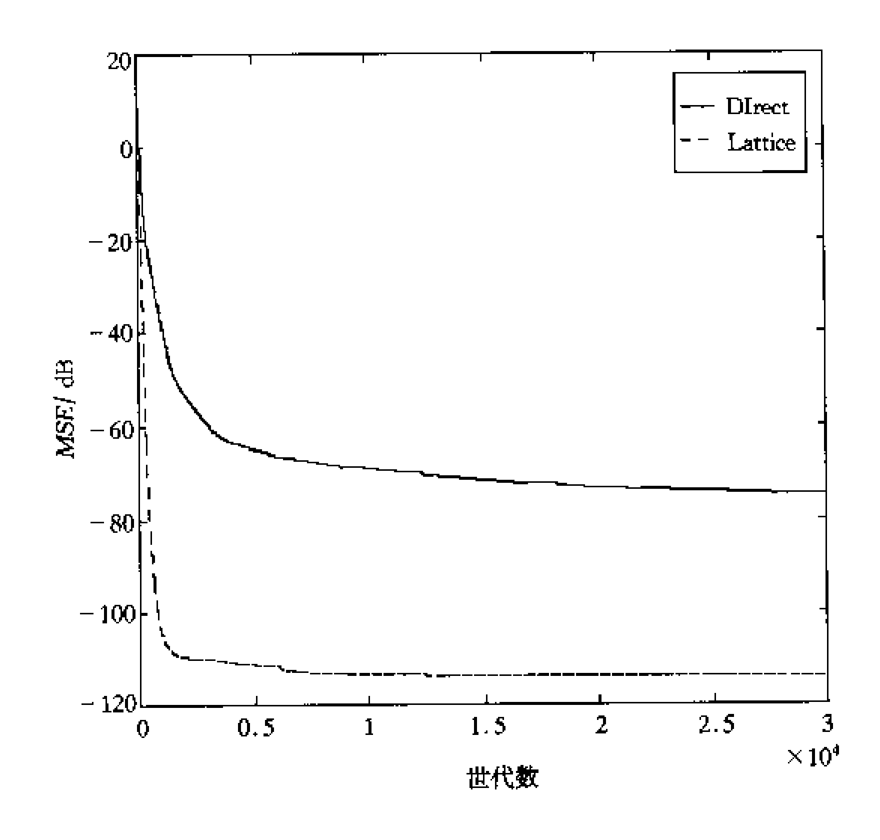


图 9.2　模型 I 的进化过程曲线

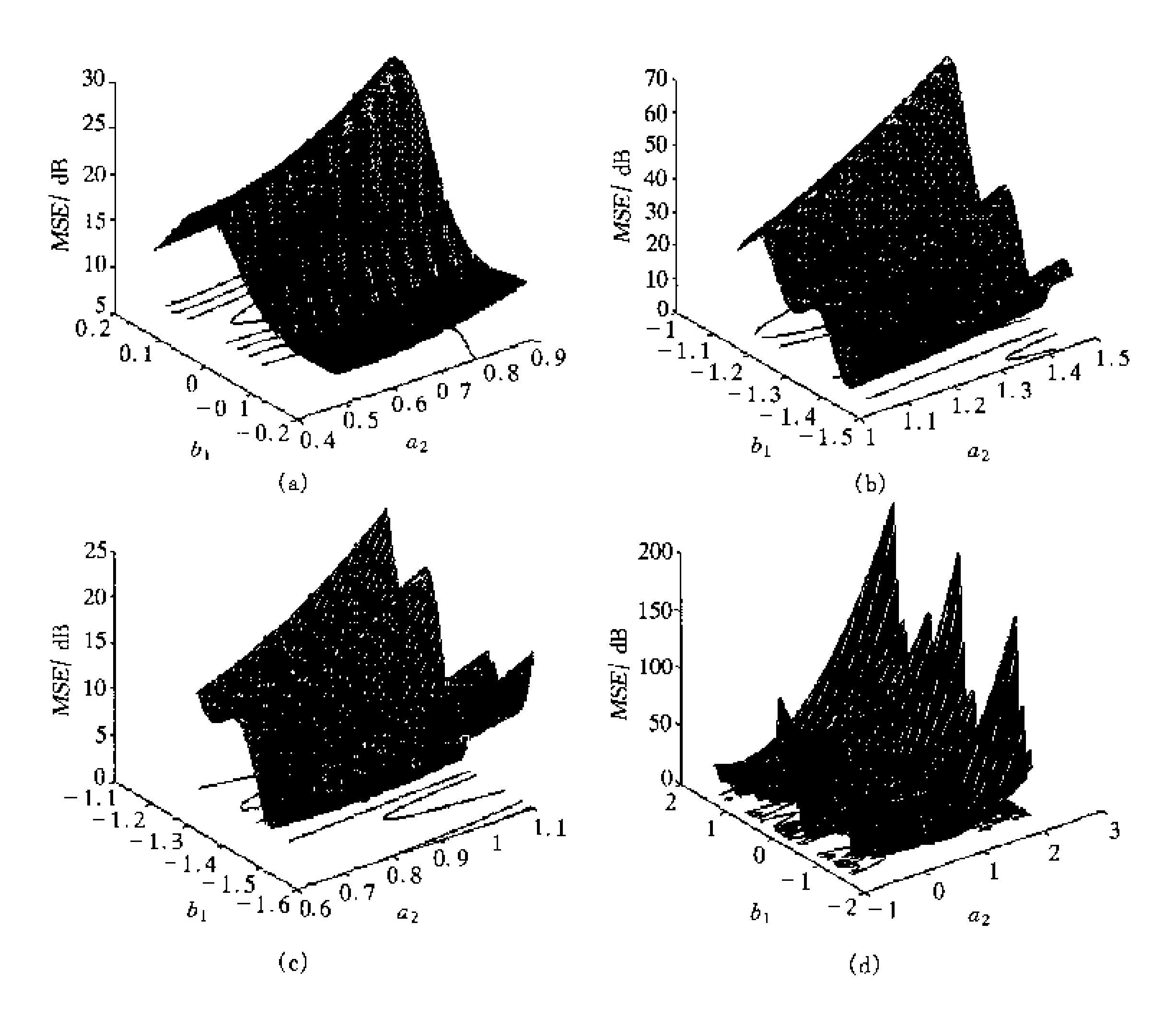


图 9.3 模型Ⅰ直接实现形式进化过程中最优个体附近的 *MSE* 曲面

(a)第 1 代； (b)第 50 代； (c)第 1 000 代； (d)最优解附近全局 *MSE* 曲面

(d)中最深的波谷处。在对 Lattice 结构形式的模拟中可以进行类似的计算和分析，图 9.4 为 Lattice 结构形式的 *MSE* 曲面的进化情况，可见 Lattice 滤波器的 *MSE* 曲面较为平坦，没有局部极小点，最优解处于一个最陡谷底。而且 Lattice 滤波器的收敛性比直接形式滤波器要快得多。

辨识实例模型Ⅱ为一个三阶系统：

$$H(z)=\frac{0.015\,4(1+3z^{-1}+3z^{-2}+z^{-3})}{1-1.99z^{-1}+1.572\,0z^{-2}-0.458\,3z^{-3}} \tag{9.11}$$

该模型有三个极点 p_1, $p_2=0.664\,7\pm0.502\,0$, $p_3=0.660\,5$，且 $|p_1|=|p_2|=0.833\,0$, $|p_3|=0.660\,5$。两极点 p_1, p_2 靠近单位圆。模型的级联和并联形式根据直接形式的一阶和二阶滤波器描述。三种实现形式的进化规划模拟计算结果如图 9.5 所示，其中以 Lattice 结构形式的性能最好，最终 *MSE* 收敛于 −69.8 dB；级联形式紧次之，最终 *MSE* 收敛于 −69.1 dB；并联形式再次之，最终 *MSE* 收敛于 −49.6 dB。图 9.6(a)～(d)为 4 种实现形式最优解附近的全局 *MSE* 曲面。同样可以发现 Lattice 滤波器误差曲面比较平坦，但缺少局部最小。级联形式滤波器误差曲面有两个相距很近的局部最优解，这种鞍点的存在造成梯度方法搜索速

图 9.4　模型 I Lattice 结构形式进化过程中最优个体附近的 *MSE* 曲面

(a)第 1 代；(b)第 50 代；(c)第 1 000 代；(d)最优解附近全局 *MSE* 曲面

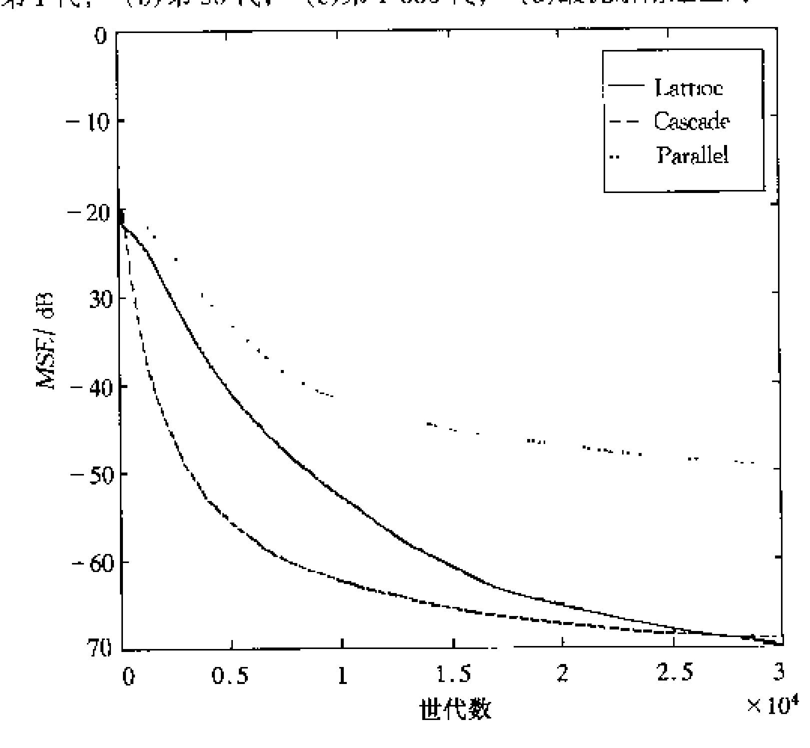


图 9.5　模型 II 的进化过程曲线

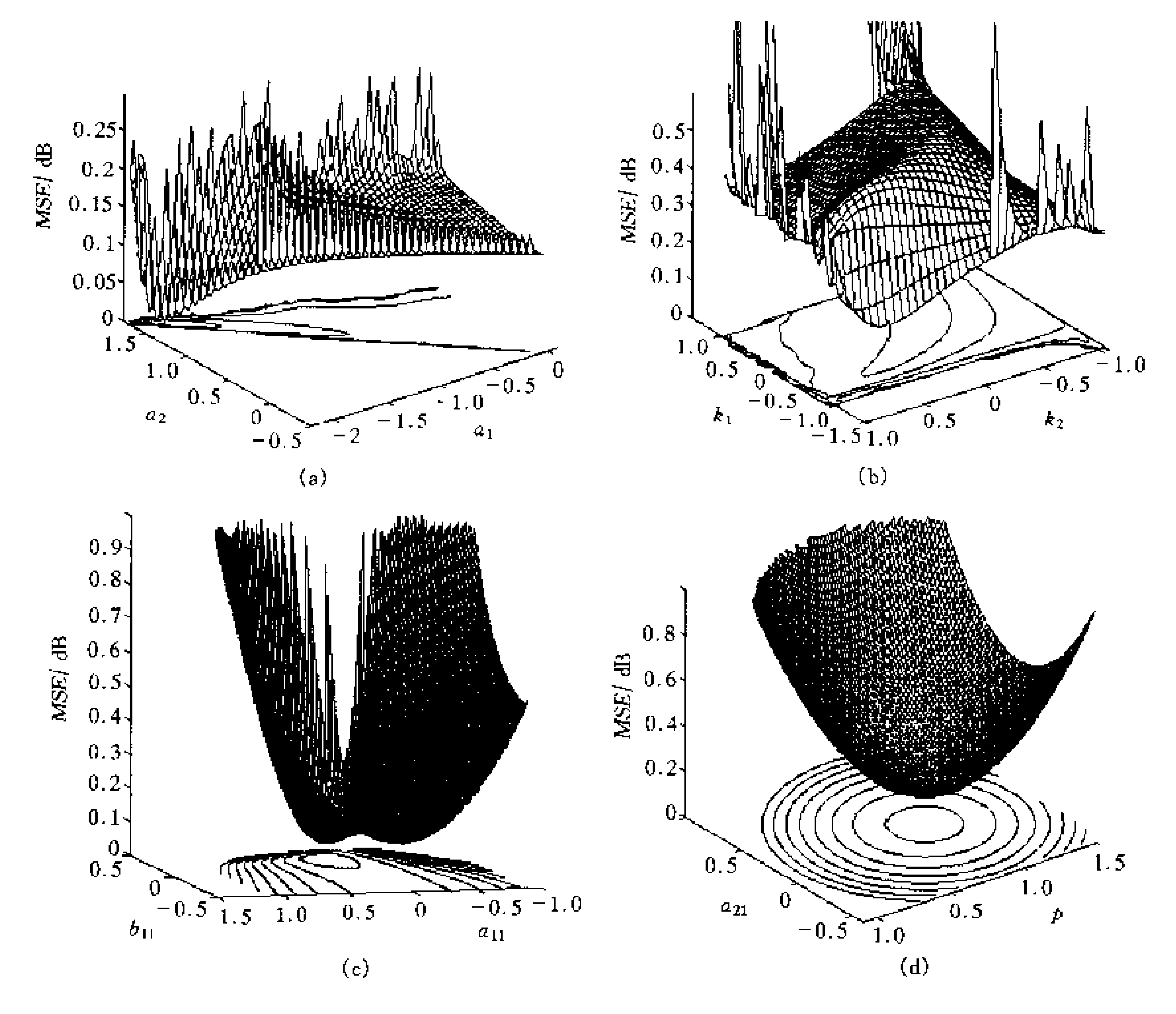


图 9.6 模型 II 最优解附近全局 *MSE* 曲面

(a)直接形式； (b)Lattice 结构形式； (c)级联形式； (d)并联形式

度大为减低，而进化规划方法则不受此影响。并联形式滤波器误差曲面似碗状，没有明显的局部最小。显然，进化规划对于并联形式滤波器的优化效率很高。

总之，进化规划应用于自适应 IIR 滤波器的设计是非常有效的，对于误差曲面鞍点的存在或有多个局部最小点场合，进化规划方法总能够找到稳定的最优解。收敛速度依赖于滤波器实现形式，也受模型的阶数影响。在多种实现形式中，Lattice 结构形式对于中短字长滤波器是较为适宜的，而级联形式更适合于长字长滤波器。低阶滤波器的 Lattice 结构形式误差曲面缺少局部最小点，高阶滤波器的级联形式误差曲面为碗状，这些场合对于进化规划方法的应用而言较为理想。进化规划方法应用于自适应滤波器设计，如果配合局部优化的方法，可以改善搜索的收敛速度。

9.2　非线性系统辨识

过去30多年来，人们对于线性、时不变和具有不确定参数的对象进行辨识和控制研究取得了很大的进展。这些研究中辨识器和控制器的结构选择和保证整个系统全局稳定的自适应调参规律的构成等，均是建立在线性系统理论基础上的。对于非线性系统的辨识和自适应控制却一直难于找到相应的数学方法。本节考虑两种非线性系统辨识方法及其遗传算法的应用。

9.2.1　GMDH方法

对于非线性系统，$\Omega=\{x_1, x_2, \cdots, x_n\}$为输入变量，$Y$为输出变量，其输入与输出的一般函数关系为：

$$Y = f(x_1, x_2, \cdots, x_n) \tag{9.11}$$

前苏联数学家伊万连科（Ivakhenko, 1970）借用生物控制论中的自组织原理，将此函数离散为Kolmoogrov-Gabor多项式：

$$Y = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} x_i x_j + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{ijk} x_i x_j x_k + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ijkl} x_i x_j x_k x_l \cdots \tag{9.12}$$

上式穷尽了输入变量自身和相互间的各种组合，因此被认为是非线性模型的完全描述，并可达到实际问题的最佳拟合。然而对于一般非线性系统想要得到此表达式的真实表达式，却远不是简单的事。为此，许多学者结合统计学、模式识别、自组织映射等理论，进行了各种拟合算法的探讨，但大多数仅限于理论上的推导。其中比较成功的是Tamura, H于80年代提出的GMDH(group method of data handling)算法及其改进算法。基本的GMDH方法见图9.7所示。

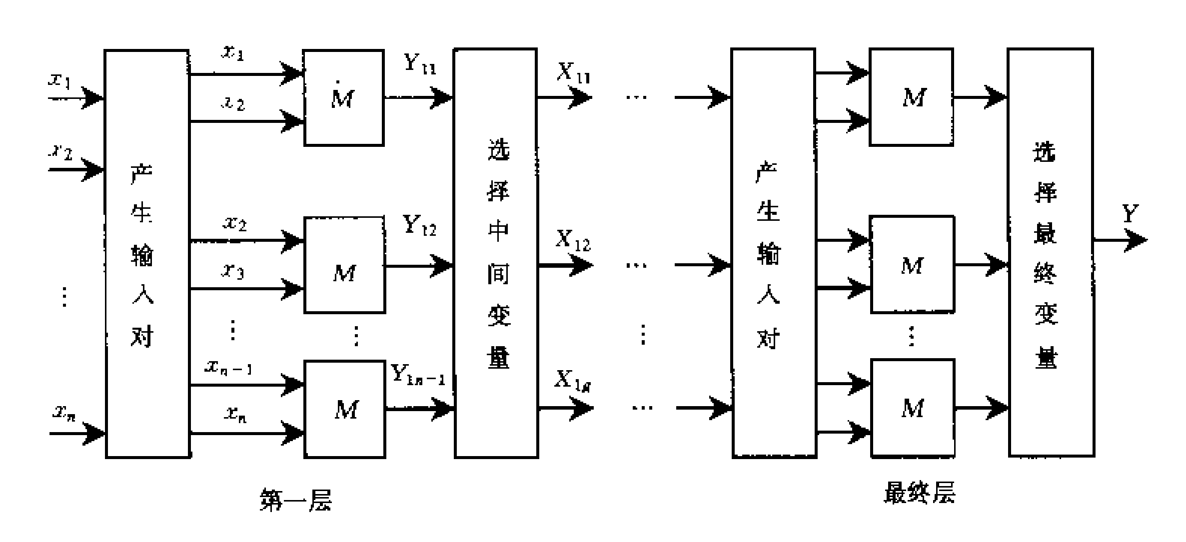


图9.7　基本的GMDH方法示意图

GMDH算法是通过多层筛选变量的办法求得模型(9.12)的近似描述。图9.7中M为部

分多项式,它是两个输入变量的完全二次多项式,即 $M=f(a_0,x_i,x_j,x_ix_j,x_i^2,x_j^2)$,它是由部分模型拟合实测数据辨识计算得到的输出;$x_{ij}$ 为中间变量,是从 Y_{ij} 中按照某种准则筛选出来的,作为下一层的输入。假定通过 N 层得到最终模型,若部分多项式 M 二阶项均被选入,则最终模型将是 2^N 阶结构的多项式。确定部分多项式结构及选择中间变量的常用准则如下:

1. 预测误差平方和(PESS)准则

$$PESS=\sum_{t=1}^{m}[Y(t)-Y'(t)]^2 \tag{9.13}$$

简化形式为:

$$PESS=\sum_{t=1}^{m}\left[\frac{Y(t)-Y'(t)}{1-x_t^{\mathrm{T}}x_t/(X^{\mathrm{T}}X)}\right]^2 \tag{9.14}$$

上式中 $Y'(t)$ 为部分多项式拟合在第 t 次输出的估计值。

$$x_t^{\mathrm{T}}=(1\ \ x_{it}\ \ x_{jt}\ \ x_{it}x_{jt}\ \ x_{it}^2\ \ x_{jt}^2),\quad i\neq j,\ t=1,2,\cdots,m$$

$$X^{\mathrm{T}}=(x_1\ \ x_2\ \ \cdots\ \ x_m)$$

2. AIC(a information criterion)准则

该准则是由日本学者赤池(Akaike)于 1973 年提出的用于过程辨识的判别准则,其判别式为:

$$AIC=m\ln S_k^2+C+2k \tag{9.15}$$

$$S_k^2=\frac{1}{m}\sum_{t=1}^{m}[Y(t)-Y'(t)]^2 \tag{9.16}$$

其中 C 为常量,k 是独立可调参数的个数。

GMDH 方法的中心问题是变量的筛选和组合,不同的变量组合辨识结果差异很大,基于一般的判别准则很容易陷入局部最优解。1991 年 Kargupta 和 Smith 等提出了遗传程序设计的方法应用于 GMDH 问题,GMDH 的变量组合非常适合于表示为一个树状结构,如图 9.8 所示。他们只采用了二次多项式构造树结构,却获得了很好的辨识结果。

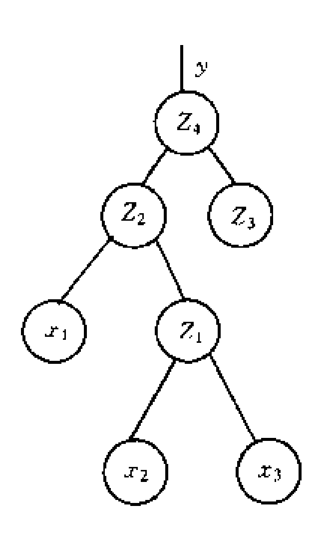


图 9.8 GMDH 中变量组合生成树

9.2.2 基于径向基函数的模糊模型辨识

近年来,非线性动态系统辨识中最常用的为径向基(radical basis function , RBF)神经网络方法。其特点是从函数逼近的角度出发,将输入直接通过非线性映射成为输出的神经元,非线性映射的输入输出之间函数关系取自径向基函数集。它非常适合于多变量函数的逼近,如果 RBF 中心点集选择得当,只需很少的神经元就可获得很好的辨识效果。

1994 年,Wienholt 将$(\mu,\lambda)-ES$ 方法应用到基于径向基函数的模糊模型辨识中。首先设定输入变量 $\boldsymbol{x}$ 为 n 元向量,输出记为标量 y。N 组输入输出数据为$\{(\boldsymbol{x}(1),y(1)),(\boldsymbol{x}(2),y(2)),\cdots,(\boldsymbol{x}(N),y(N))\}$。输入变量各分量及输出变量分别用 M_1 和 M_2 个模糊数定义。这里模糊数采用下列高斯型隶属函数描述:

$$\mu_{ij}(x_i)=\exp\left(-\frac{1}{\sigma_{ij}^2}(x_i-c_{ij})^2\right),\quad j=1,\cdots,M_1 \tag{9.17}$$

$$\mu_j(y)=\exp\left(-\frac{1}{\sigma_j^2}(y-c_j)^2\right),\quad j=1,\cdots,M_2 \tag{9.18}$$

如图9.9所示为变量 x_i 的隶属函数的示例。

模糊模型的输出 y' 按下式计算：

$$y' = \frac{\sum_{i=1}^{K} A_i c_i}{\sum_{i=1}^{K} A_i} \tag{9.19}$$

其中，$A_i = \prod_{j=1}^{n} \mu_{ij}(x)$。

采用下面的辨识算法进行隶属函数中参数学习、规则的选择，以及参数调整后辨识模型的建立。

第1步 设定初始候选规则集，对各个变量定义等间隔的高斯模糊数。其中，

$$\sigma_i^2 = \left(4 \cdot \frac{c_{i,j+1} - c_{i,j}}{6}\right)^2 \tag{9.20}$$

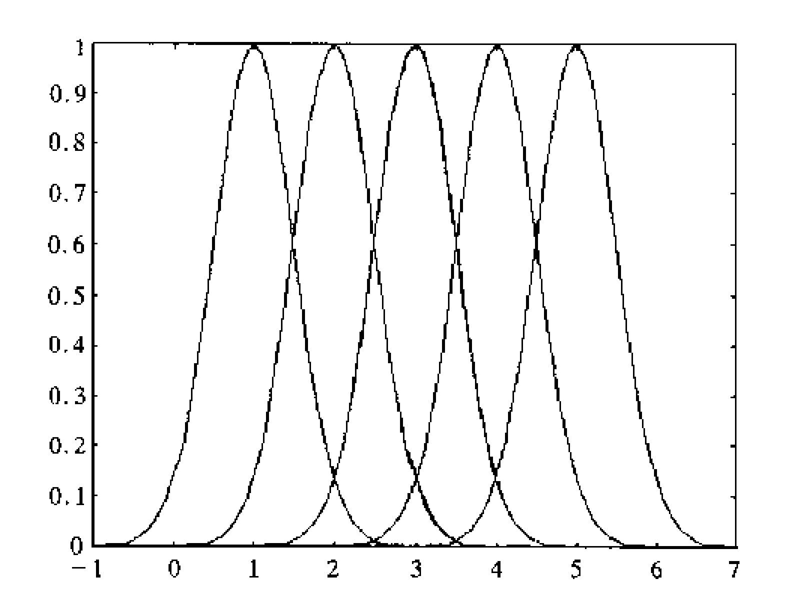


图9.9 隶属函数示例

第2步 按进化策略方法进化计算优化RBF集的中心参数。

第3步 根据样本数据获得规则集，按相容性和完备性要求大幅度减少规则数。

第4步 规则后件部模糊数中心参数 $c_i(i=1,2,\cdots,K)$ 按最速下降法学习。

在第2步中 *ES* 方法的具体做法是，将所有 *RBF* 集的中心参数 $\{c_{ij}\}$ 排列为一个向量 c，c 的分量个数 q 为 $M_1 n + M_2$，作为进化的个体描述为 (c, s)，其中 $s=\{s_{ij}\}$ 为与 $\{c_{ij}\}$ 一一对应的随机数。进化初始值可以取相同的数值 $s_{ij}=(1/10)\sigma_j$。对于个体 $i=1,2,\cdots,\lambda$ 一代的运算操作如下：

① 在参数集 $\Gamma=\{1.5, 1, 1/1.5\}$ 中等概率选择变异参数 γ。

② 从父个体集 $\{s_{kj}^p\}$ 中随机选择 ρ 个个体，按下式计算子个体的 s_{kj}^0：

$$s_{ij}^0 = \gamma \frac{1}{\rho} \sum_{k=1}^{\rho} s_{kj}^p \tag{9.21}$$

然后，选择一个个体 i^*，对其实施变异操作，计算公式如下：

$$c_{ij}^0 = c_{i^*j}^p + \frac{s_{ij}^0}{\sqrt{q}} z_{ij} \tag{9.22}$$

上式中，$z_{ij} \sim N(0,1)$（均值为0，方差为1的正态分布）。

③ 从 λ 个个体中选择评价值较小的 μ 个个体作为下一代的父个体，进入下一代的进化。个体的评价值按下式计算：

$$E = \sum_{k=1}^{N} (y(k) - y'(k))^2 \tag{9.23}$$

约定一定的进化代数，作为ES的终止条件。在进化末代获得最优的RBF集的中心参数，然后进入第3步。

在第3步中，对于一组输入输出样本 $(x_1(k), x_2(k), \cdots, x_n(k), y(k))$，根据式(9.17)和(9.18)求出 x 的各分量和 y 值对应的最大隶属度所处的模糊数编号 j，这样构成由 x 分量对

应的模糊数作为前件、y 值对应的模糊数作为后件的一条候补规则。记 $x(k)$ 对应的模糊数编号分别为 $I_1(k), I_2(k), \cdots, I_2(k)$，而 $y(k)$ 对应的模糊数编号为 $J(k)$，则有

$$I_i(k) = \{j : \max_j \mu_{ij}(x_i(k))\}, \quad i = 1, 2, \cdots, n \tag{9.24}$$

$$J_i(k) = \{j : \max_j \mu_j(y(k))\} \tag{9.25}$$

对 N 组数据进行类似的处理，可以获得 N 条规则。由于存在许多前件相同而后件不同的矛盾规则，需要进行筛选。其方法是计算这些规则的前件部分隶属度与后件部分隶属度之积，保留隶属度积较大的规则。此外，需要剔除对大部分数据不适合的规则。最终获得一定数目 K 的规则集。

根据 K 条模糊规则可以计算模糊模型的输出，按式(9.19)计算模型的输出值，其中 A_i 为第 i 条规则的隶属度，c_i 为规则 i 的后件部模糊数中心。

9.3 控制系统设计自动化和高性能实现

现代计算技术的两项重要研究成果在控制工程中的应用是：其一为基于遗传算法的进化技术用于控制器设计的自动化和系统辨识；其二为基于“舒张阵列”(systolic & wavefront array)和“传算机”(transputer)的并行处理技术用于控制系统的高速实时实现。本节将侧重研讨系统辨识和控制系统设计问题的统一描述方法以及存在的困难，并使用遗传算法来系统地解决这些问题。

随着控制技术的发展，现代系统工程师面对实际的工程控制问题往往需要在多个可行方案中作出选择，这已经成为司空见惯的事了。例如，对于一个控制系统模型，控制器的参数优化设计便是这方面的典型问题。许多控制系统的辨识建模和设计问题都可以归结到含噪声的多峰值空间内多参数系统优化的应用。常规优化方法的成功应用更多地依赖于目标函数的“良好表现”，而实际上对于较复杂的控制对象而言，这只是一相情愿的事。常规的分析方法和数值方法由于受到初始估计的影响，只有较小的搜索空间，因此很可能陷入局部最优。

我们知道，遗传算法最成功之处莫过于解决困难的优化问题。大量实践表明，遗传算法在从参数优化入手进行控制系统离线设计方面非常有效。Krishnakumar 和 Goldberg(1992)以及 Bramlette 和 Cousin(1989)采用遗传算法在飞行器控制器结构的应用研究中发现，遗传算法在函数计算方面花的时间比 LQR 或 Pwell 增益集设计方法要少得多。Varsek 等(1993)将进化计算应用到控制器结构的选择和调整中；Gleghorn(1989)等采用遗传算法研究了机器人在稳定和非稳定环境下的路径规划问题。随着遗传算法应用需求的不断增长，一些计算机辅助控制系统设计(CACSD)软件包也提供了基于遗传算法的高精度建模和设计接口，作为系统离线模拟和设计的高级技术。

9.3.1 控制系统设计方法以及存在的困难

不失一般性，我们考虑一个二阶线性定常(LTI)连续系统，其传递函数模型如下：

$$G(s) = \frac{a_2 s^2 + a_1 s + a_0}{b_2 s^2 + b_1 s + b_0} \tag{9.26}$$

该模型的辨识问题即建模问题，其中心任务是最优估计 6 个参数，即 a_i 和 b_i ($i \in \{0, 1, 2\}$)。而基于该模型控制系统的设计问题，该模型表示为一个控制器，控制器设计的中心任务

是优化模型的参数以获得最佳的设计性能指标。通常，控制系统的建模和设计问题对应于一个多参数优化问题。对于建模问题而言，目标函数可以是最小化误差；对于设计问题而言，目标函数可以是一些反映设计要求的性能指标的极大或极小(当然，还包括可能的实际系统约束)，它们主要包括：

① 极好的瞬时响应性能，包括上升时间、超调量和调节时间；

② 极好的稳态响应性能，包括稳态余差；

③ 一定的稳定性裕量；

④ 较好的鲁棒性能，包括抗干扰性和参数灵敏性的要求。

一般地，常规优化设计方法应用于控制系统建模和设计时，存在以下几个方面的困难：

(1) **多目标问题** 在许多工程实践中，通常需要考虑多个控制目标，在均衡多个相互矛盾的目标达成中选择折衷解或者偏解，而这些控制目标很难通过加权的方法归化为单个综合目标来处理。

(2) **目标函数问题** 常规数值优化方法基本上属于牛顿型或斐波那契(Fibonacci)型梯度方法，一般要求所谓的“表现良好”，如连续可微以及相对平滑的等势面等等。

(3) **约束问题** 常规优化方法处理实际工业应用中的硬约束条件非常棘手，这些约束条件包括直接的域约束，如参数范围、固定关系等，以及非直接不等式约束，如电压和电流的极限。

(4) **多峰值问题** 串行搜索导向通常容易趋于局部极值点，而常规并行方法缺少在并行搜索点之间有效的信息交换机制，因此，对多峰值问题并没有很大的改善。

(5) **先验性(A-Priority)问题** 一般很难在设计过程中融入设计师的先验知识和经验，而这对于复杂控制对象往往是很重要的。

此外，对于一个成功的控制系统 CAD，还需要面临来自于系统复杂性、设计质量和精度、设计成本和速度、设计可靠性和安全性等许多方面的挑战。而现有多数的 CAD 系统在这些困难和挑战中步履维艰。例如，设计者在使用一个 CAD 软件包进行设计时，一般需要进行启发式模拟分析，他们首先会输入一些先验性的控制器参数，然后借助软件进行模拟和手工分析，如果模拟性能指标不能达到设计标准，可以随机地或根据经验修改参数值进行一系列重复模拟实验，直到获得所谓的“满意”设计方案。很明显，这样常规的设计过程是半自动化的，设计参数组合的随意性使得设计结果难以预计，而且所谓的“满意”解不能保证达到系统最优性能。如果依靠枚举法进行参数全空间的搜索，实现自动寻优，对于参数较少且(精度要求)较低时是可行的，但对参数较多且精度要求较高时，现有的计算工具是不能胜任的。如对于(9.26)式对应的控制器设计问题，假设每个参数按精度要求在取值范围内离散处理 128 个数值，则正规化处理后存在 4 个参数需要辨识，参数全部组合有 128^4 个，如果一个控制器评价计算时间平均为 0.1 s，则计算所有组合的时间为 0.1×128^4 s(为 10 个月)，这显然是难以承受的。

9.3.2 遗传算法应用于控制系统建模和设计

由于遗传算法既不需要借助评价函数的求导信息，也不依赖初始解的估计，而是依靠模仿来自于自然界生物进化的搜索机制，在全局解空间中并行搜索最优解，因此，它比常规方法更适合解决控制系统设计的诸多困难。迄今为止的大量应用表明，遗传算法适用范围很广，从简单的 PID 控制器、线性控制器设计问题，到最优控制、自适应控制、鲁棒控制、滑模控制、模糊

控制、神经网络控制等工程控制问题，都有许多成功应用的范例，因而是一种极具潜力的方法。

下面给出一个直流伺服线性定常系统辨识和设计的应用实例。设定开环伺服电机系统模型微分方程式为(9.27)式。

$$\frac{d^2\omega}{dt^2}+(\frac{JR+LB}{LJ})\frac{d\omega}{dt}+(\frac{RB}{LJ})\omega=(\frac{K_T}{LJ})v_{in} \tag{9.27}$$

这里 $v_{in}\in[-5V,5V]$为输入控制电压，作为一种间接约束，K_T 为转矩常量(Nm/A)，R 为电机线圈阻抗(Ω)，L 为线圈感应系数(H)，B 为机轴摩擦系数(N·m·s)，J 为载荷的惯性矩(kg·m²)。上述模型的传递函数形式为式(9.26)，其中 $a_2=0$, $b_2=1$。

遗传算法应用于该模型的辨识，染色体的编码方法采用二进制编码，4 个参数变量映射为 28 位二进制串，其中每个变量对应 7 位二进制串，表示变量范围内有 128 个可能值。种群的大小为 50，选择操作采用排序选择，交叉率为 0.6，变异率为 0.01。模型的输入激励采用单位阶跃。模型输出与样本输出之间的误差作为个体评价测度。按照个体的 e_{sys}排序序位 k 计算个体的适应度，计算公式为式(9.29)。

$$e_{sys}(P_i)=\sum_{j=1}^{N}|\omega_j-\hat{\omega}_j| \tag{9.28}$$

$$f(k)=\frac{2k-1}{\sum_{k=1}^{50}(2k-1)}=\frac{2k-1}{250} \tag{9.29}$$

模拟运算的终止条件为种群平均适应度改善量在 7% 内。经模拟实算，获得以下辨识参数的最优值：

$$\frac{JR+LB}{LJ}=39.142,\qquad \frac{RB}{LJ}=86.186,\qquad \frac{K_T}{LJ}=0.054$$

对于上述辨识模型 $G(s)$对应的控制对象系统，同样可用遗传算法设计控制器 $H(s)$，控制器的优劣可根据控制系统的性能评价而定。通常对于一个已知模型，设计者预先需要选择一种控制器，如 PID 型控制器、相位超前控制器和相位滞后控制器、基于极点配置的状态反馈控制器、最优 LQG 控制器、H_∞控制器等等。从基于性能评价和自动设计的要求出发，控制器可以规范到一个统一的形式，从而避免预先确定其中的一种。

控制器的参数优化与辨识模型的遗传优化过程相类似，所不同的是适应度评价需要综合控制系统的性能指标，有时还需要考虑系统的约束条件。如式(9.30)所示的就是一种综合反映系统稳态和瞬时响应的简单误差函数。

$$e_{design}=\sum_{j=1}^{N}[|e_j|+|\Delta e_j|]j \tag{9.30}$$

其中，e_j 为时刻 j 的闭环误差，Δe_j 为时刻 j 误差改变量，这种线性加权形式比较好的综合反映了上升时间、超调量和稳定性能，避免了渐进稳定性或收敛性的分析。应该说，控制器的性能综合评价不存在统一的形式，依赖于具体的应用问题。

将遗传算法应用于上述直流伺服电机辨识模型的控制器设计，获得最优控制器的传递函数 $H(s)$为(9.31)式，对经过辨识建模和设计的系统模拟计算，输入 60 r/mim 的阶跃激励，系统的响应曲线见图 9.10 所示。

$$H(s)=\frac{19.27s^2+121.66s+108.84}{s^2+72.77s+0.14} \tag{9.31}$$

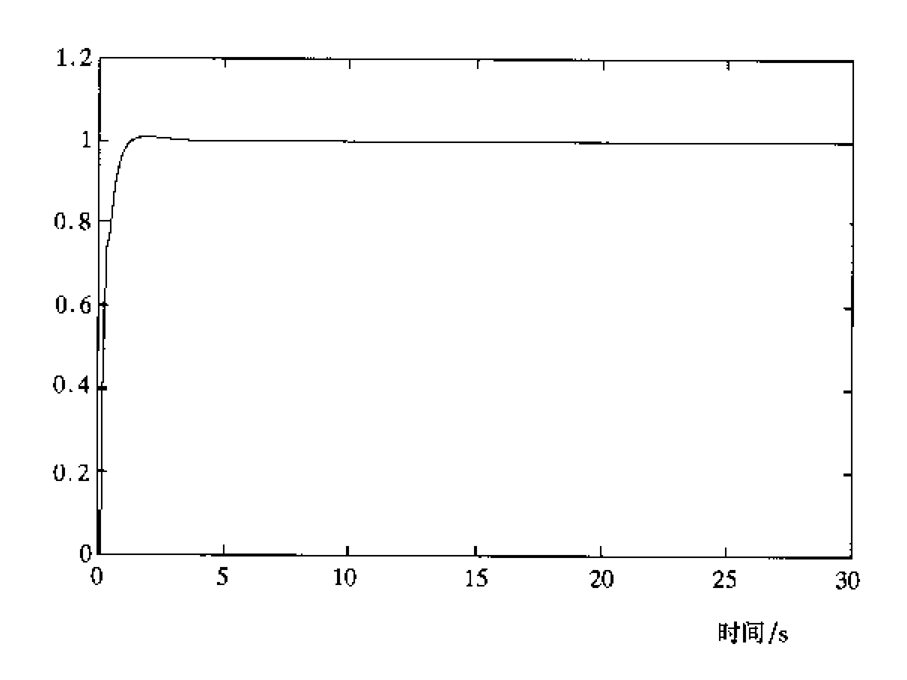


图 9.10　经遗传优化的直流伺服电机控制系统的阶跃响应

9.3.3　控制系统的高性能并行实现

在现代计算技术中，计算智能建立了认知科学和复杂系统的理论研究框架，而并行处理为系统的高性能计算、高精度实时实现提供了可能，这对于包含计算智能应用的高级控制系统而言，尤为重要。一般的实时控制系统需要牺牲系统在处理动态性和复杂性方面的能力，采取简单的控制算法，满足了实时的要求，却可能损失了系统的性能。与高性能信号并行处理系统相比，反馈控制系统并行处理的实时要求更加严格，除了信号吞吐量要求外，处理时间延迟是需要同时加以考虑的问题，因为偏长的处理延迟会影响整个系统的稳定性相位裕量。

目前，基于 VLSI 的舒张阵列并行处理机能够满足嵌入式控制系统在信号吞吐量和处理时间延迟方面的设计要求。图 9.11 为最简单的一种并行处理机结构。该结构实际上是一种实现在 A100 并行 DSP 上的半舒张阵列机，由 T212 传算机将控制器的系数值和输出误差计算值传递给 A100 阵列机。

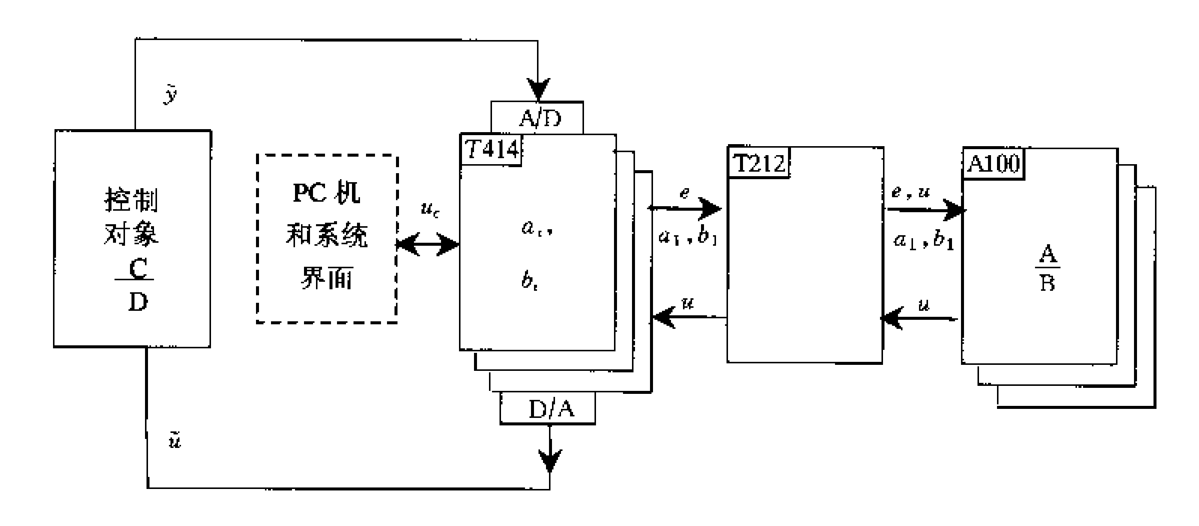


图 9.11　基于 Transputer/A100 和遗传算法的自适应控制系统异质结构

作为一种新型自适应控制系统,将自动设计和系统实现融为一体,是向自治控制系统(autonomous control system)发展的重要方向。除了 A100 阵列机外,INMOS 传算机负责遗传算法应用于系统辨识和系统自动设计工作。图 9.12 表示了这一自适应控制系统的结构,它属于一种 MIMD 系统。值得注意的是,控制器的优化设计和系数更新远比控制信号的产生和传递要慢得多,但一般能够达到 47 kHz 的速度,这对于多数工程应用已经足够了。

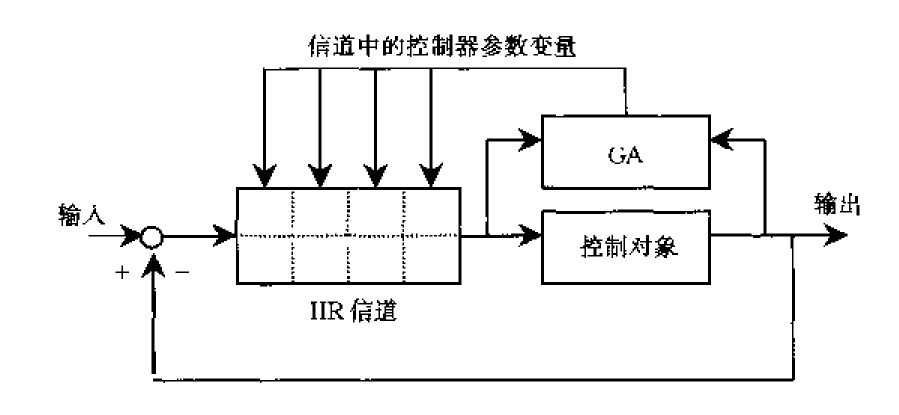


图 9.12 基于遗传算法的自适应控制系统结构

9.3.4 遗传算法应用于控制系统设计的多目标优化方法

控制领域的问题只需要单目标优化的情况很少,相反,需要同时满足许多互相匹配的目标。通常,我们可以用数学规划的方法求出 Pareto 最优解集。有些方法(如 e-约束法,加权和法以及目标规划法)往往需要一些通常难以获得的权重和目标值的精确表达式。如果获得了设计目标之间的折衷等势面,这些方法通常需要迭代求解。此外,非线性规划方法不能很好地处理多峰值和函数不连续性,因此可能只能找到局部解。用以求解多目标最优化问题的遗传算法称为多目标遗传算法(multi-objective genetic algorithms, MOGA)(详见 6.4 节内容)。

1995 年,英国谢费尔德大学的 C. M. Fonseca, P. J. Fleming 等人将 MOGA 应用于飞马座燃气涡轮机低压线轴速度控制器的优化设计中,他们结合设计决策者在目标、约束以及其优先级方面的偏好信息,提出了根据关系算子进行 Pareto 最优性排序,计算个体的适应度。还引入分享机制和本地算子以避免遗传漂移现象,保持种群的多样性。并且按渐进的策略交互偏好信息,使得最优解成为实际上的满意解。在模拟实算中,考虑了以下几个方面的问题:

① 非线性的控制系统模型用 Simulink 模拟实现,并给定初始条件和控制器参数设置,遗传算法程序用 Matlab 语言编写,因此系统模拟可以在一个计算环境中执行。

② 控制器参数按格雷编码(grey encoding)方法映射为染色体编码,对于 5 个控制器参数,染色体长度为 70 位,种群大小为 80。

③ 交叉操作采用两点缩小代理交叉,变异操作采用一般的二进制变异方法。

④ 优化目标上升时间、调节时间、超调量和输出误差,其中上升时间 t_r 定义为达到 70% 最终输出的时间,要求 $t_r \leqslant 0.59$ s;调节时间 t_s 定义为达到 ±10% 最终输出范围的时间,要求 $t_s \leqslant 1.08$ s;超调量 OS,要求 OS≤10%;输出误差 err,要求 $err \leqslant 10\%$。

⑤ 在遗传算法实算过程中,设计决策者将直到当前代的所有非支配解存储起来,构成设计的先验性知识(a-priority knowledge),决策者根据 Pareto 最优性的概念和先验性知识表达自

己的偏好信息，设定新的目标值，反馈给遗传算法的评价过程。遗传算法在新一代运算中根据改变的目标值，聚焦到决策者感兴趣的若干搜索区域。一般与设计者的交互可在相隔一定演化代数之后进行，经过这样演化实算可以保证得到一组满意解。

图9.13(a)～(c)分别为经过第40代（初始目标值）、第40代（新目标）、第60代（新目标值）的目标折衷图，基本上认为第60代时获得了一组令人满意的最优解。图9.13(d)为模拟获得的一组满意解的系统阶跃响应曲线。

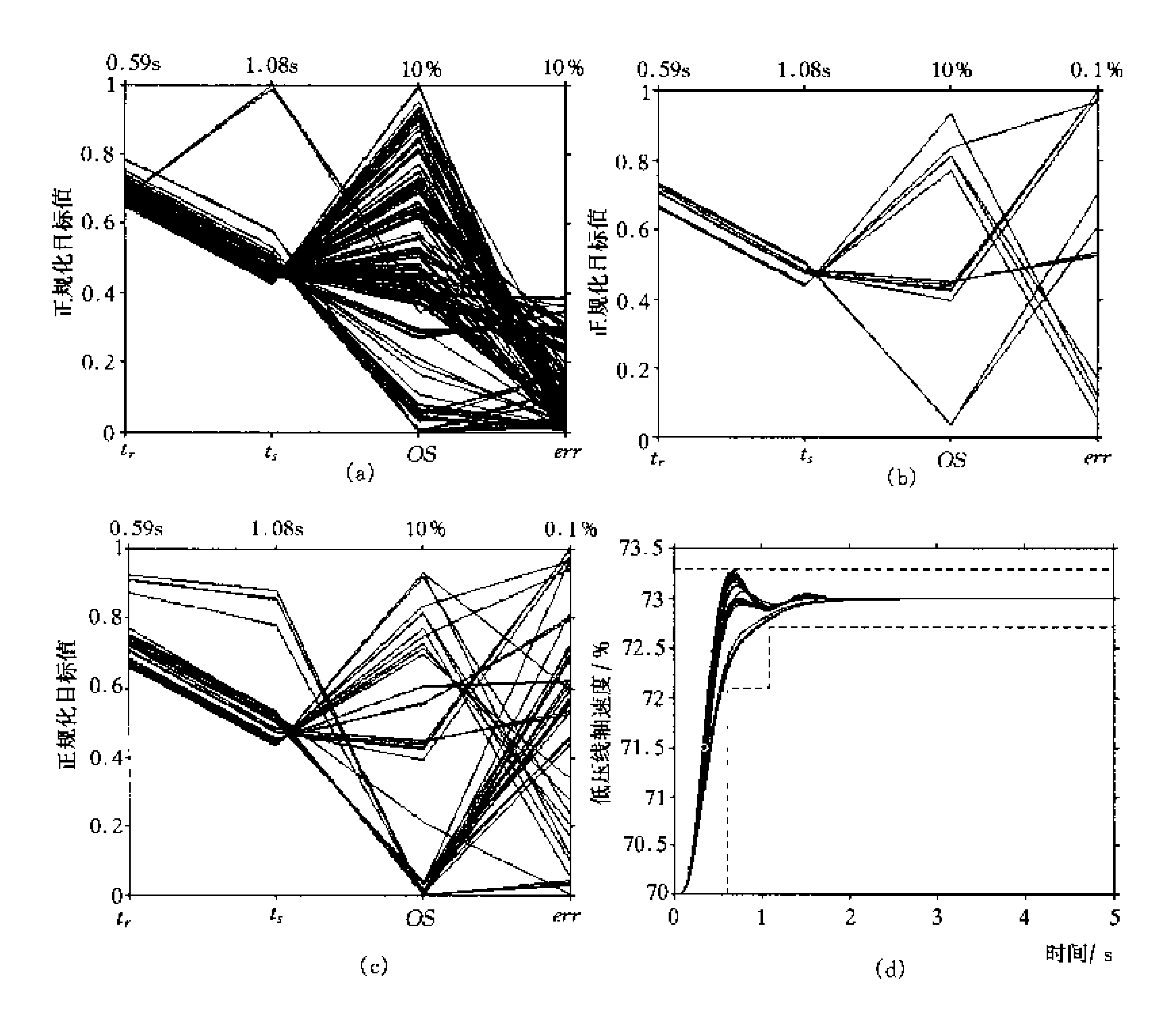


图9.13　燃气涡轮机低压线轴速度控制器的遗传算法优化实算结果

(a) 第40代目标折衷图（初始目标值）；　(b) 第40代目标折衷图（新目标值）；

(c) 第60代目标折衷图（新目标值）；　(d) 第60代获得的一组满意解阶跃响应

9.4　基于遗传算法优化的模糊控制

模糊逻辑控制器（fuzzy logic controller, FLC）特别适用于无法得到准确数学模型、多输入、具有不确定因素、非线性系统的控制，其主要优点在于直观易懂、易于利用专家知识、具有较好的鲁棒性和适应能力，并且开发成本低、周期短，因而广泛应用于工业过程、家用电器和运动控

制领域中。在过去的模糊控制器设计中,模糊规则的获取和控制器参数的调整都没有系统的方法,主要依靠控制专家的经验和设计者的反复试验。随着系统复杂度的提高,直观经验越来越难以获得,而且往往表达不清楚,难以直接利用,因此寻求一种具有自动设计和优化的方法已成为目前亟待解决的问题。

9.4.1 模糊控制器设计方法的分析

常规模糊系统是由模糊控制器和被控对象组成的单回路反馈控制系统。它以系统输出 y 与设定值 r 的偏差 e 以及偏差变化率 ce 为输入变量,以执行机构的输出 u 为输出变量。模糊逻辑控制器一般由四部分组成:模糊化环节、规则库、推理机和反模糊化环节,如图 9.14 所示。

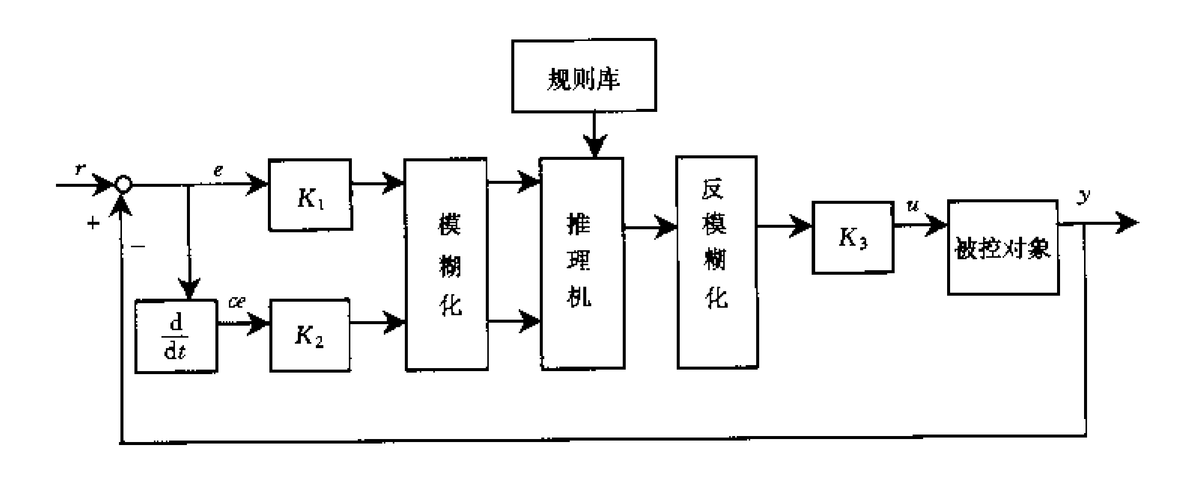


图 9.14 模糊控制系统结构

模糊化环节的功能先通过参数 K_1 和 K_2 将输入变量 e 和 ce 的值规格化为标准论域上的精确值,然后通过模糊化算法转化为描述输入变量大小的语言值。规则库是一组语言控制规则,即一系列以 IF－THEN 形式表示的模糊条件判断句,其前件描述系统所处的状态,后件描述控制输出。若选取偏差 e 及偏差变化率 ce 作为模糊控制器的输入,控制动作作为输出,可建立如下形式的控制规则:

$$R_i:\quad \text{IF } e \text{ IS } A_i \text{ AND } ce \text{ IS } B_i \text{ Then } u \text{ IS } C_i \tag{9.32}$$

式中:R_i 代表规则库中的第 i 条规则;e, ce, u 分别为描述偏差、偏差变化率和控制输出的语言变量;A_i, B_i, C_i 分别是相应语言变量的语言值。模糊推理实质是模拟人的推理过程,当输入为某一组具体的条件时,利用已有的模糊控制规则,得出相应的控制输出。反模糊化环节是将模糊推理得到的控制输出语言值转化为驱动执行机构动作的精确值。

下面简单说明一下模糊控制器的设计过程。

(1) **确定模糊控制器的结构** 模糊控制器的结构指的是哪些变量是它的输入变量,哪些变量是它的输出变量。由于模糊控制规则是总结操作人员控制经验得到的,所以,选择的变量应当是操作人员能观察的变量。在工业过程控制和家用电器控制中,一般选择被控对象的输出变量与设定值的偏差及偏差的变化作为模糊控制器的输入变量,而把控制量作为模糊控制器的输出变量。

(2) **选择描述输入、输出变量的模糊语言** 模糊语言即模糊状态或模糊集合。模糊语言选择得多,说明变量可以用比较多的模糊状态来描述。这样,制定规则时比较灵活,规则也比较细致,但可能使规则变得比较复杂。所以选择模糊语言时不一定是选择的数量越多就越好,

而需要兼顾简单性和灵活性两个方面。一般，每个变量以选用 2～10 个模糊状态为宜。根据经验，控制进度要求高的场合选 7～8 个模糊状态，例如，负大(NB)、负中(NM)、负小(NS)、零(ZE)、正小(PS)、正中(PM)、正大(PB)，或 负大(NB)、负中(NM)、负小(NS)、负零(NZ)、正零(PZ)、正小(PS)、正中(PM)、正大(PB)，控制精度要求一般的场合可选择 5 个模糊状态，例如，负大(NB)、负小(NS)、零(ZE)、正小(PS)、正大(PB)。

(3) **确定控制规则**　控制规则是模糊控制器的核心。将操作人员大量成功的控制策略分析归纳后应给出输入、输出变量的模糊状态描述，就得到控制规则。在 PI 型模糊控制器设计过程中，一般将所有的控制规则汇总成控制状态表(lookup table)。表 9.1 为某一控制器的控制状态表。为了保证控制过程的快速性和稳定性，控制决策一般以两种形式给出，即：当偏差很大时，控制决策为满输出和零输出，也就是说以绝对形式给出；当偏差在一定范围(论域范围)时，以控制量的增量形式给出，即：

$$u(kT) = u[(k-1)T] + \Delta u(kT) \tag{9.33}$$

表 9.1　控制规则表

e	NB	NS	ZE	PS	PB
NB			PM	PB	PB
PS		ZE	PS	PM	PB
ZE	NM	NS	ZE	PS	PM
NS	NB	NM	NS	ZE	
NB	NB	NB	NM		

模糊控制规则的生成大致有以下四种方法，即：

① 根据专家经验或过程控制知识生成；

② 根据过程模糊建模生成；

③ 根据对手工控制操作的系统观察和测量生成；

④ 根据学习算法生成。

(4) **确定变量论域和比例因子**　将变量的实际变化范围划分成若干等级，把这些等级的全体作为变量的论域。等级划分得多，一般说控制效果会更好。但等级划分得过多，势必增加设计工作量，同时控制表也会更复杂。一般控制品质要求高的场合，可划分成 13～15 级；要求一般的场合可分成 6～8 级。如某一变量论域为 U，划分为 13 级，则 $U=\{-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6\}$。输入变量模糊化时，引入输入变量比例因子 K_1 和 K_2，将实际变化范围内的输入值转换为论域范围内有关等级值。假设 $\{-e_m,\cdots,e_m\}$ 和 $\{-e_{cm},\cdots,e_{cm}\}$ 分别为关于偏差和偏差变化的论域，$\{-e_n,\cdots,e_n\}$ 和 $\{-e_{cn},\cdots,e_{cn}\}$ 分别为偏差和偏差变化的实际变化范围，则有：

$$K_1 = \frac{e_m}{e_n}, \quad K_2 = \frac{e_{cm}}{e_{cn}} \tag{9.34}$$

确切输入量和比例因子相乘，取其距离相近的等级值，即成为论域中的元素。

同样，经模糊推理和决策得到的控制量也不是实际驱动执行机构的确切控制量，而是关于

控制量的论域中的一个等级。它和实际控制量之间也有一个比例关系，因此，类似地引入输出比例因子 K_3，通过比例因子 K_3 将论域中的等级值转换成确切的控制量。设 $\{-u_m, \cdots, u_m\}$ 为关于控制量的论域，$\{-u_n, \cdots, u_n\}$ 为控制量的实际变化范围，则有：

$$K_3 = \frac{u_n}{u_m} \tag{9.35}$$

(5) **定义模糊状态的隶属函数**　隶属函数的形状和参数选择是否符合客观实际，将直接影响控制效果。一般常选择对称三角形、对称梯形、正态型隶属函数。其参数可以先初步确定，再通过试验逐步修改。隶属函数的形状大致可分为高分辨力和低分辨力两类，一般在大偏差范围内采用低分辨力的模糊集合，而在小偏差情况下采用高分辨力的模糊集合。在定义各模糊状态的隶属函数时，要考虑到它对论域的覆盖程度，以及各模糊集合之间的相互影响。图 9.15 给出了某一模糊控制器的输入、输出变量论域内模糊状态的隶属函数定义示例。

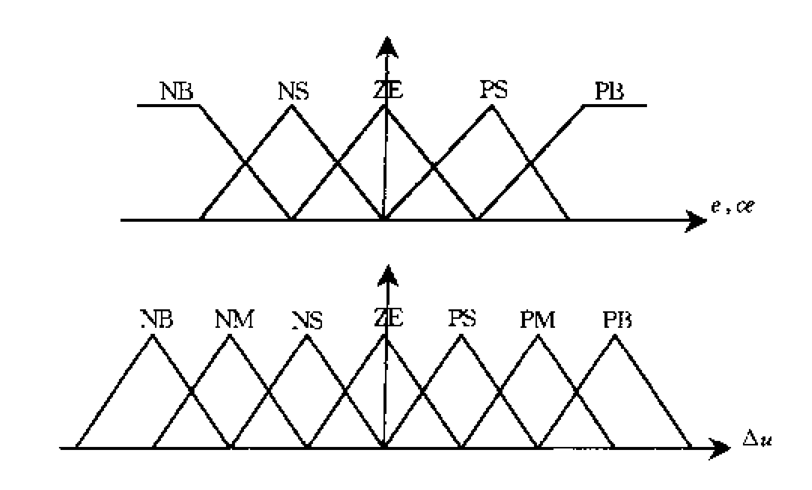


图 9.15　$e, ce, \Delta u$ 模糊化语言变量

(6) **确定模糊推理关系**　模糊推理关系是根据控制规则对控制器对应的输入、输出的模糊集合进行直接运算后再进行并运算的结果，它代表了控制器的输入输出关系。比较典型的方法有 Zadeh 方法、Baldwin 方法、Tsukamoto 方法、Yager 方法和 Mizumoto 方法。两输入一输出控制器的模糊关系 $\tilde{R}$ 可表示为：

$$\tilde{R} = \bigcup_{i,j}(\tilde{E}_i \times \tilde{E}_{cj} \times \tilde{u}_{ij}) \tag{9.35}$$

式中，$\tilde{E}_i$ 为偏差 e 的第 i 个模糊状态，$\tilde{E}_{cj}$ 为偏差变化 ce 的第 j 个模糊状态，$\tilde{u}_{ij}$ 为控制规则中对应第 i 个偏差模糊状态和第 j 个偏差变化模糊状态时输出量的模糊状态。

求得模糊推理关系 $\tilde{R}$ 后，运用模糊集合的合成运算，可以推得相对于输入模糊信号的输出模糊信号。二输入一输出控制器的合成运算为：

$$\tilde{u} = (\tilde{E} \times \tilde{E}_c) \circ \tilde{R} \tag{9.36}$$

(7) **反模糊化**　也称模糊判决，通过模糊推理得到的结果是一个模糊集合，但在实际使用中，必须判决一个相对能代表这个模糊集合的单值，常用的判决方法有最大隶属度法、重心法、系数加权平均法和隶属度限幅元素平均法。

模糊控制器分为基于领域的 FLC(domain - based FLC)和基于规则的 FLC(rule - based FLC)。如图 9.16 所示，常规的 FLC 大多是基于领域的，其特点是具有覆盖输入输出空间的

全局的模糊划分,对每一输入输出划分的组合均有一模糊规则对应,也即具有完整的规则集。基于规则的 FLC 一般没有完整的规则集,规则的数目是可变的,更重要的是,隶属函数是局部定义的,依赖于相应的规则。基于规则的 FLC 由于具有可变数目的规则,能适应于从简单到复杂的各类控制对象,具有很强的可伸缩性,尤其是适合于多输入的复杂的控制任务。与基于领域的 FLC 相比,基于规则的 FLC 还有一个显著的优点,是它可以不用预先定义隶属函数的个数。基于规则的 FLC 大都需要预先确定模糊划分(隶属函数)的个数,以此来设计规则集。规则库中可能的最大规则数为各输入变量模糊划分的乘积,若划分太细,规则数目急剧增长,使 FLC 结构变得复杂,增加计算的负担;若个数太少,又不能满足 FLC 性能上的要求,可能造成未定义的盲区。在基于规则的 FLC 中,规则的定义不需要全局的隶属函数,因而规则的数目是可以自由变动的。其缺点是可靠性不能得到充分的保证,因为局部定义的隶属函数不能保证覆盖全部的输入空间,造成在输入空间中某些区域 FLC 没有相应的规则与之对应。另外,局部定义的模糊集隶属函数也削弱了模糊集的语义特性。

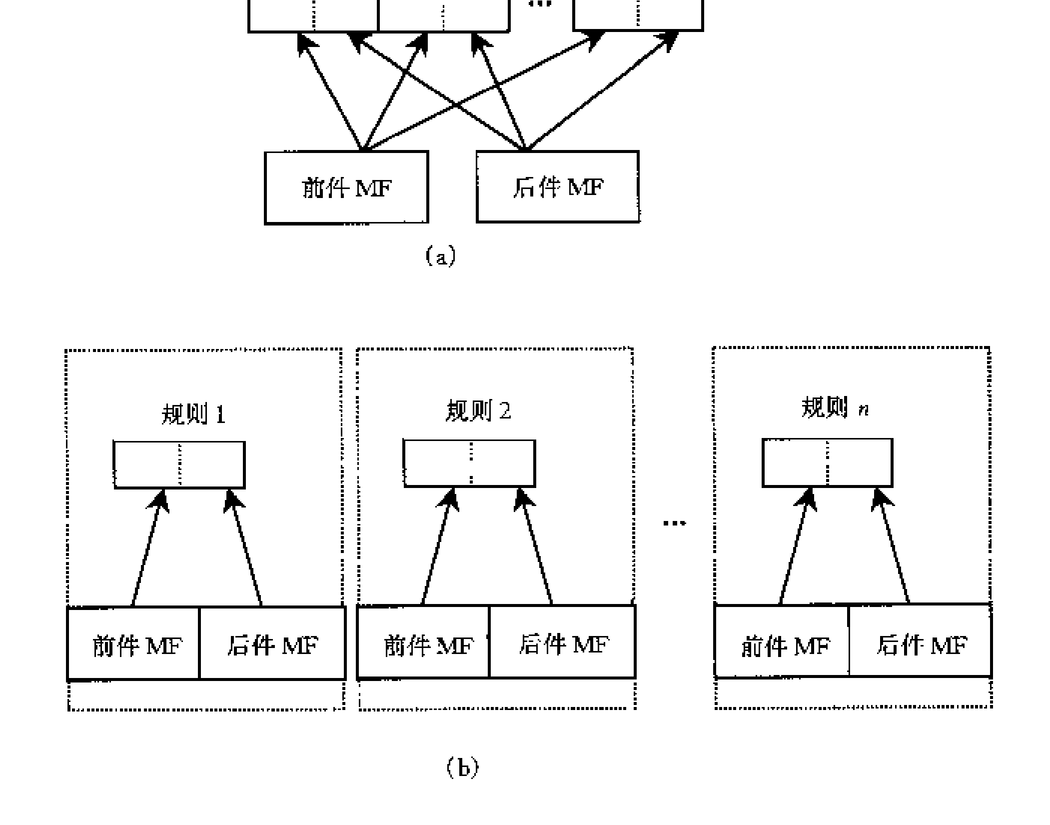


图 9.16　两种 FLC 结构比较(MF:隶属函数)
(a) 基于领域的 FLC;　(b) 基于规则的 FLC

模糊控制器设计过程存在两个主要问题:一是模糊控制规则的选取和优化。由于缺乏有效的知识获取手段,模糊控制规则的获取主要依靠经验;二是模糊变量的隶属函数的正确选取。在模糊控制规则确定的情况下,模糊控制器的性能由模糊变量各模糊状态的隶属函数来确定。这是多参数寻优问题,在一般情况下无法获得全局最优。

近年来，人们已经引入神经网络、梯度法等算法解决以上问题。例如，对模糊规则的优化主要采用神经网络的方法，利用神经网络的自学习、自组织特性，将神经网络和模糊系统融合，使模糊系统能够自动地从经验中获取模糊规则。但由于神经网络存在网络结构和网络规模较为复杂以及学习收敛性差等问题，很难获得令人满意的结果。而梯度法的速度较快，但要求性能评价函数连续可微，且容易陷入局部最优，因而受到一定的限制。

9.4.2 基于遗传算法的模糊控制器优化设计方法

考虑到模糊控制器的优化涉及到大范围、多参数、复杂和不连续的搜索表面，人们自然想到用遗传算法来进行优化。遗传算法应用于模糊控制器的优化设计是非常适合的，遗传算法的运行仅由适应度数值驱动而不需要被优化对象的局部信息。此外，优化模糊控制器正好符合遗传算法的所谓"积木块"(building block)假设，积木块指长度较短的、性能较好的基因(gene)片段。模糊集的划分有相当的重叠，这使得相邻的隶属函数之间能产生强烈的相互作用和组合效应，而远离的隶属函数之间则没有或只有极小的相互影响，因此，若按顺序将这些模糊集排列起来形成遗传算法的个体，良好的模糊集能容易地形成积木块；模糊规则集也有类似的性质。考虑一个二输入一输出的规则表，不失一般性，假定任一输入都激活规则表中的四条规则，那么这四条规则是相邻的，且排列成正方形。若规则采用矩阵类型的编码，这四条规则就可能构成一个积木块，即使规则采用一维编码，也可以构成两组由两条规则组成的积木块。

模糊控制器可以看作一种学习分类器系统。在第8章中，我们介绍了遗传算法应用于机器学习的两种主要方法：密歇根方法和匹兹堡方法。将这两种方法应用于模糊控制器的学习，前者将每一条规则作为一个个体，整个种群代表一个系统(控制器)，系统性能的优劣由整个种群来决定。遗传算于作用于个体之间或单个个体上。个体的好坏由个体所获得的"分数"(credit)来衡量的，分数是由相应的分配算法来确定的。该分配算法是实现密歇根方法的关键，遗传算子作用于个体水平上，而性能评价则是基于多条规则或整个种群的，这不可避免地会造成规则之间的冲突，这种冲突只能由各规则的得分来调和。那么如何将种群的评价合理地分配给各个体？这是密歇根方法面临的主要问题。后者将整个规则集(整个控制器)作为一个个体，多个不同的控制器形成一个种群，遗传算子作用于个体水平，性能评价也在个体上进行，这就解决了个体之间冲突的问题。匹兹堡方法解决了密歇根方法的困难，也带来了新的问题。首先是计算量明显增加，因为需要评价多个控制器。同时，个体的适应度值(性能)同样难以反馈到单条规则上，这就是所谓的"强化信息的带宽较窄"，从而难以提高单条规则的性能，使得遗传算法的学习效率低。用遗传算法设计模糊控制器，从方法看，一般采用密歇根方法和匹兹堡方法的改进，如Nagoya方法。Nagoya方法将个体分为若干部分，每一部分都包含数目大致相等的规则，将突变算子施加给第一部分若干次，分别计算出各次突变后个体的适应度，选出其中最好的一次替换原个体中相应的部分。对个体的其余部分也作类似的处理，直至各部分都得到更新。这时产生一个新的性能有相当提高的个体，对群体中的每个个体都执行上述步骤后，再进行选择、交叉等经典操作。

从模糊控制器的类型看，既有基于领域的FLC，也有基于规则的FLC，还有模糊神经网络，模糊神经网络具有全局的隶属函数，可以看作是基于领域的FLC。常规的模糊控制器是基于领域的，具有思路直观、可靠性强、规则易于理解的优点。由于其规则和隶属函数的个数事先确定，遗传优化时的个体的编码长度是一定的，也合乎经典遗传算法的要求，无须对遗传

算法作大的改动。个体长度固定也有助于估计搜索空间的规模，把握学习的进程。

用遗传算法优化模糊控制器时，优化的主要对象是 FLC 的隶属函数和规则集。在基于领域的 FLC 中，隶属函数是全局定义的，它与规则是相独立的。遗传算法在设计这类 FLC 时，有下面几种不同的类型：

① 已知模糊控制规则集，优化隶属函数；

② 已知隶属函数，优化规则集；

③ 规则集与隶属函数由 GA 分阶段优化；

④ 同时优化规则集和隶属函数。

最早进行这方面研究的是美国矿业局的 C. L. Karr，他用遗传算法来微调倒摆控制器隶属函数的位置、形状等参数，结果表明 GA 优化后的隶属函数远远优于手工设计的。

后来 P. Thrift 用遗传算法设计一个二输入一输出的模糊规则集，控制对象是一个在一维无摩擦轨道上运行的小车，目的是让小车停在轨道上的指定位置(如中点)，输入变量是小车的位移和速度，输出变量是作用在小车上的力。输入输出变量均划分为 5 个三角形的模糊集，依次为负中(NM)，负小(NS)，零(Z)，正小(PS)，正中(PM)，这样就构成一个 5×5 的规则表，表中的每一项从(0, 1, 2, 3, 4, 5)(分别对应于 NM, NS, Z, PS, PM, - ；其中“ - ”代表控制器无动作)中任意取值。将这个二维表按一定顺序展开成一维，就形成遗传算法的个体。用某一个体的规则控制小车的运行，所得到的结果作为该个体的适应度值。Thrift 所使用的交叉算子为经典的两点交叉，变异算子定义为选中的突变基因变为其相邻的模糊集或“ - ”。小车模拟运行的时间为 10 s，时间间隔为 0.02 s。个体的适应度值定义为 $500-T$，其中 T 为小车在 25 次不同初始状态模拟中停在中点的平均用时。种群大小为 31，经过 100 代后，得出的控制器优于优化“bang - bang”控制。

同时优化隶属函数和控制规则的工作，最早可见于 M. A. Lee 和 H. Takagi 以一个简化的倒摆系统为例，用遗传算法设计一个二输入一输出的 T - S 型模糊逻辑控制器。算法设计中个体分为两部分：输入变量的隶属函数参数和控制规则的后件参数，每个输入变量划分为 10 个三角形的模糊集，每个模糊集用 3 个参数表示，共有 $10\times10=100$ 条规则，每条规则有三个后件参数，因此每个个体有 $2\times10\times3+10\times10\times3=360$ 个参数。采用二进制编码，每个参数用 8 个二进制位表示，故个体总长为 2 880 位，探索空间是十分巨大的。后来为了减少规则的数目，对适应度函数增加一个惩罚项减少个体中活动的规则数，最后得到只有 4 条规则的 FLC。

由于同时优化隶属函数和规则大大增加了搜索的复杂性，Kinzel 提出了分阶段优化的方法，遗传学习的前期主要优化控制器结构，后期主要是参数的微调，尤其是隶属函数参数的调整。其步骤如下：

① 根据已有的知识产生一“好”的规则集；

② 对输入输出空间进行手工划分，建立隶属函数，并以此为基础，用遗传算法优化规则集；

③ 在前两步的基础上，用遗传算法微调隶属函数参数。

一般认为，基于领域的 FLC 的性能最终取决于隶属函数的个数，因为 FLC 实质上是一种插值器，插值基函数是输入隶属函数。隶属函数的个数对 FLC 的性能影响更大，一旦隶属函数的个数确定后，FLC 可能的最大规则数也随之确定，换言之，隶属函数的个数的决定表现在对最大规则数的限制上。有时由于规则数偏少，一味地调整规则前后件参数是无济于事的；若

规则数目太多，又会增加学习的负担。问题在于模拟进化的过程中自主地改变规则的个数。基于领域的 FLC 无论采用哪种优化方法，输入隶属函数的个数总是预先确定的，据此才能构造出规则，这极大地限制了对 FLC 结构的优化，而优化工作几乎集中在 FLC 的参数优化上。

基于规则的 FLC 最大的优点是伸缩性很好，大大减轻了对专家知识的依赖，尤其适合于复杂的、了解很少的控制对象。其缺点也很突出，首先是适当的交叉算子难以定义，在交叉后容易出现"结构/功能"问题；其次，局部定义的隶属函数不能保证覆盖整个输入空间，有可能出现不稳定的情况，而且隶属函数在很大程度上丧失了语义，所得到的规则可理解性较差。

总而言之，基于遗传优化的模糊控制器设计方法具有许多其他方法难以比拟的优越性，遗传算法可以优化 FLC 从结构到参数数值的各个方面，可以较少地依赖先验知识，可以没有训练数据对，易于嵌入设计者的知识提高学习效率，易于与其他方法结合使用。目前这方面的研究远未成熟，但大量的研究表明，下面的几个问题值得注意：

(1) **遗传算法的改进** FLC 的优化可以看作参数优化问题，遗传算法在其他参数优化领域使用的改进策略，大多可以用于优化 FLC，如精英(elitist)策略，两点或多点交叉，自适应概率算子等。

(2) **嵌入专家知识** 为了提高学习效率，专家的知识和经验应该贯穿进化过程的始终。初始化种群中包含一个或多个由专家手工设计的个体；在进化过程中应根据经验对参数的取值范围进行限制，如模糊集"正大"的最高点应该位于"正中"的右边，两模糊集交叉区域不应超过各自的最高点位置。

(3) **合适的编码方式** 经典的遗传算法采用二进制编码，但越来越多的研究表明实值编码在数值优化方面有更高的精度和效率。基于遗传程序设计的树结构层次编码以及 DNA 编码方法能够很好地适合于多输入多输出、控制规律复杂不清的场合。

(4) **适合的适应度函数** 选择适应度函数被认为是遗传算法中最具主观性的环节。适应度函数必须反映设计者对 FLC 的性能和结构复杂性两方面多目标的要求。性能包括多个目标，如平衡点附近的精度、到达平衡点的时间、超调等；结构复杂性包括隶属函数个数和规则条数。这些目标之间往往是互相矛盾的，多目标遗传算法是很好解决的方法。

(5) **结合局部优化的方法** FLC 的优化体现在两个层次上，即结构优化和参数数值优化。经验表明，作为全局优化器的遗传算法在数值优化中，接近最优点的效率不高。解决这个问题的方法很多，如在进化过程中结合局部优化，如共轭梯度下降法、模糊神经网误差反传法、模拟退火法等等。

下面介绍一种从 Nagoya 方法发展而来的新方法，它在编码机制以及局部改善机制方面提供了新的思路。

9.4.3 基于 DNA 编码与伪细菌遗传算法相结合的模糊控制器的控制规则学习

日本名古屋大学电子信息工程系的市桥等继提出著名的 Nagoya 方法之后，于 1997 年提出了一种新的模糊控制器控制规则学习方法。该方法受启于生物学中的基因表达原理和微生物进化机制，一方面引入 DNA 编码解决染色体长度可变的问题以及相应的遗传操作算子设计困难，另一方面利用 Nagoya 方法中伪细菌遗传算法(Pseudo - Bacterial Genetic Algorithm, PBGA)的局部改善机制，很好地解决了目前模糊控制器自动设计中面临的困难。

1. DNA 编码方法

根据生物学的知识，核酸是重要的生物大分子，是一切生物的遗传物质，担负着生命信息

的储存和传递的作用，核酸的基本结构单元是核苷酸(nucleotide)。核苷酸由碱基、戊糖和磷酸三部分组成，根据所含戊糖种类不同分为脱氧核糖核酸(DNA)和核糖核酸(RNA)。DNA由四种碱基成分腺嘌呤(adenine A)、鸟嘌呤(guanine G)、胞嘧啶(cyanine C)、胸腺嘧啶(thymine T)构成。RNA的碱基构成与DNA不同之处在于：不存在胸腺嘧啶，而代之以尿嘧啶(uracil U)。RNA主要有三种：信使RNA(mRNA)，核糖体(rRNA)和转运RNA(tRNA)。DNA分子由两条多核苷酸长链组成的螺旋结构组成，两链的碱基相互以氢键相连配对，A总是与T配对，G总是与C配对。一个DNA分子的碱基对(base pair)数目是很多的，一般有几千到几万个，多至数百万个。如果一个DNA分子有100对碱基，这100个碱基对在这个分子上的排列就有4^{100}种。最简单的病毒约有5 000个碱基对，而人的46个染色体的DNA，估计有40亿个碱基对。这说明DNA储存了无穷的遗传信息。

从DNA上携带的信息到蛋白质的最后生成，这一过程称为基因表达。如图9.17所示，首先，DNA的编码被抄写到信使RNA(mRNA)上，称为转录，然后负责传送物质的转运RNA(tRNA)将氨基酸运送到核糖体。核糖体一面读mRNA上的编码信息，一面按照编码顺序将相应的氨基酸连在一起，这个过程称为翻译。此外，RNA中的碱基U取代DNA中碱基T，多余的部分被切除，称之为拼接(splicing)。当核糖体翻译完一条mRNA时，就产生一条具有特

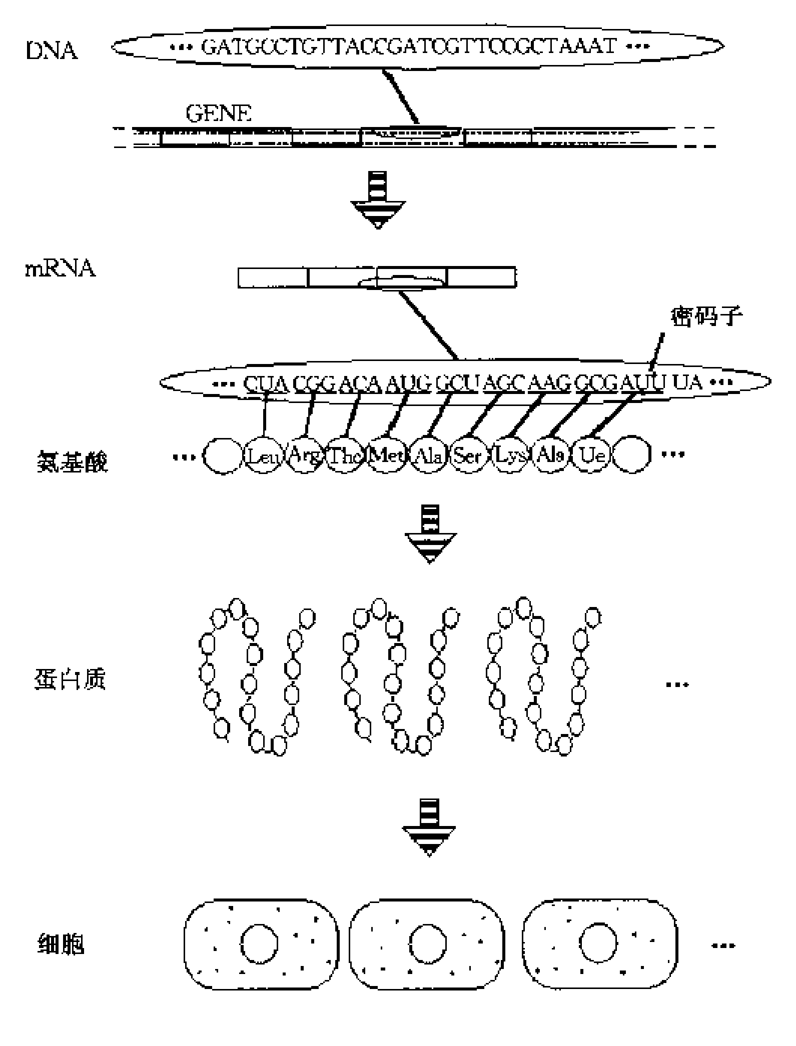


图9.17 生物基因表达示意图

定顺序的氨基酸多肽链，一条或多条多肽链再形成蛋白质，最后由蛋白质组成细胞。从 DNA 到蛋白质的遗传信息传递方向为 DNA→RNA→蛋白质。基因转录过程中，双链 DNA 转录出单链 mRNA，mRNA 是蛋白质合成的直接模板，蛋白质由 20 种氨基酸以不同方式排列而成，mRNA 与蛋白质之间的关系是通过遗传密码翻译实现的。遗传密码阐明了支配 mRNA 分子中 4 种核苷酸的线性序列同由它编码的蛋白质中 20 种氨基酸线性顺序之间的关系。mRNA 上每 3 个相邻核苷酸翻译成蛋白质多肽链中的一个氨基酸，即 mRNA 上每 3 个相邻核苷酸组成一个三联体，编码一种氨基酸，称之为密码子(codon)，如 UAG，UAA，UGA 等，总共有 $4^3=64$ 种不同的密码子，如表 9.2 所示。遗传密码是所有密码子的总和。

表 9.2 遗传密码中的密码子

第 1 位碱基	第 2 位碱基 U	第 2 位碱基 C	第 2 位碱基 A	第 2 位碱基 G	第 3 位碱基
U	UUU 苯丙氨酸	UCU	UAU 酪氨酸	UGU 半胱氨酸	U
	UUC (Phe)	UCC 丝氨酸	UAC (Tyr)	UGC (Cys)	C
	UUA 亮氨酸	UCA (Ser)	UAA 终止信号	UGA 终止信号	G
	UUG (Leu)	UCG	UAG (STOP)	UGG 色氨酸	A
C	CUU	CCU	CAU 组氨酸	CGU	U
	CUC 亮氨酸	CCC 脯氨酸	CAC (His)	CGC 精氨酸	C
	CUA (Leu)	CCA (Pro)	CAA 谷氨酰胺	CGA (Arg)	G
	CUG	CCG	CAG (Gln)	CGG	A
A	AUU 异亮氨酸	ACU	AAU 天冬酰胺	AGU 丝氨酸	U
	AUC (Ile)	ACC 苏氨酸	AAC(Asn)	AGC (Ser)	C
	AUA 甲硫氨酸	ACA (Thr)	AAA 赖氨酸	AGA 精氨酸	G
	AUG (Met)	ACG	AAG (Lys)	AGG (Arg)	A
G	GUU	GCU	GAU 天冬氨酸	GGU	U
	GUC 缬氨酸	GCC 丙氨酸	GAC(Asp)	GGC 甘氨酸	C
	GUA (Val)	GCA (Ala)	GAA 谷氨酸	GGA (Gly)	G
	GUG	GCG	GAG (Glu)	GGG	A

图 9.18 显示了模糊控制规则的 DNA 编码方法的过程，基本类似于生物 DNA 编码。常规的遗传算法编码一般无冗余部分，考虑到生物 DNA 的拼接，这里采用一种冗余编码方法。一个染色体编码由四个碱基 A，G，C，T 构成，染色体编码中有许多冗余部分，经过拼接，完成人工 RNA 的合成。在 DNA 翻译为 RNA 时，每个密码子对应一种氨基酸。与生物氨基酸不同的是，每个人工氨基酸可以有几种含义，一个基因片段的含义根据氨基酸的组合而定。实际上，一个人工氨基酸可以被理解为一个输入变量或一个模糊状态等等，这样几个人工氨基酸组成一个基因片段，对应于一条模糊规则，而所有 DNA 片段映射为控制规则集。图 9.19 表示了一个 DNA 染色体示例，图中一个基因片段从密码子 ATG 开始，终止于密码子 TAG，从而将 DNA 中的密码子翻译为人工氨基酸序列，如 Tyr，Thr，…。基因片段的起点偏移几个碱基位置，又可译出相互重叠的基因片段，如图 9.20 所示。

基于 DNA 编码方法的遗传操作设计比较方便。图 9.21(a)为一个单点交叉情形。个体 1 中包含基因片段 GENE1 和 GENE2，个体 2 包含基因片段 GENE3，GENE4 和 GENE5。两个

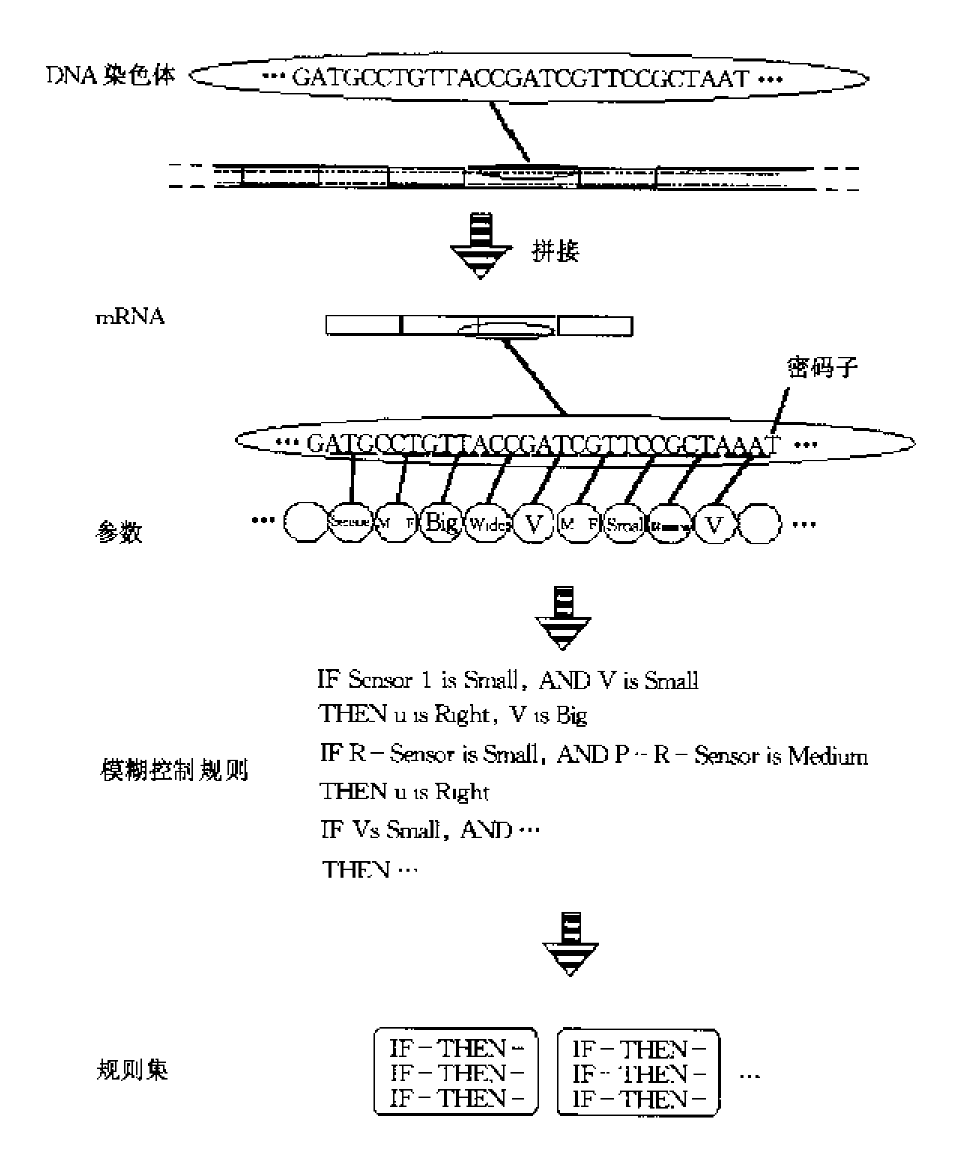


图 9.18 模糊控制规则的DNA编码



图 9.19 DNA染色体翻译机制示意图

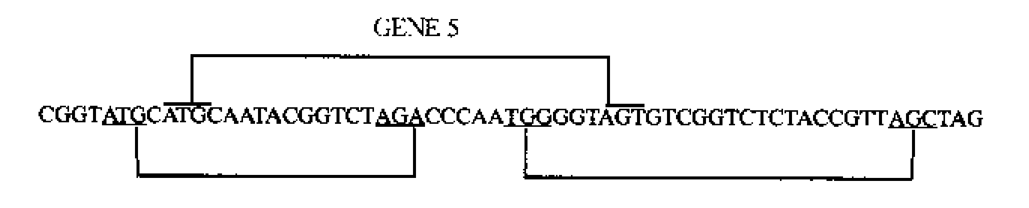


图 9.20 基因片段的重叠

个体编码串在交叉点部分右侧互换，产生两个新个体，新个体中的基因片段因此发生了变化，出现了新的基因片段 GENE 2′，GENE 4′ 和 GENE5′。交叉点的位置可随机选择，交叉的结果因交叉点位置不同可以有很多差异。至于变异操作，如图 9.21(b)所示，随机选择个体编码中一个碱基换成同类的另一个碱基，从而生成新的变异个体。如图中的个体编码中一个碱基 T 被选择变异为 G，结果生成的个体包含新的基因片段 GENE1′ 和 GENE 7。

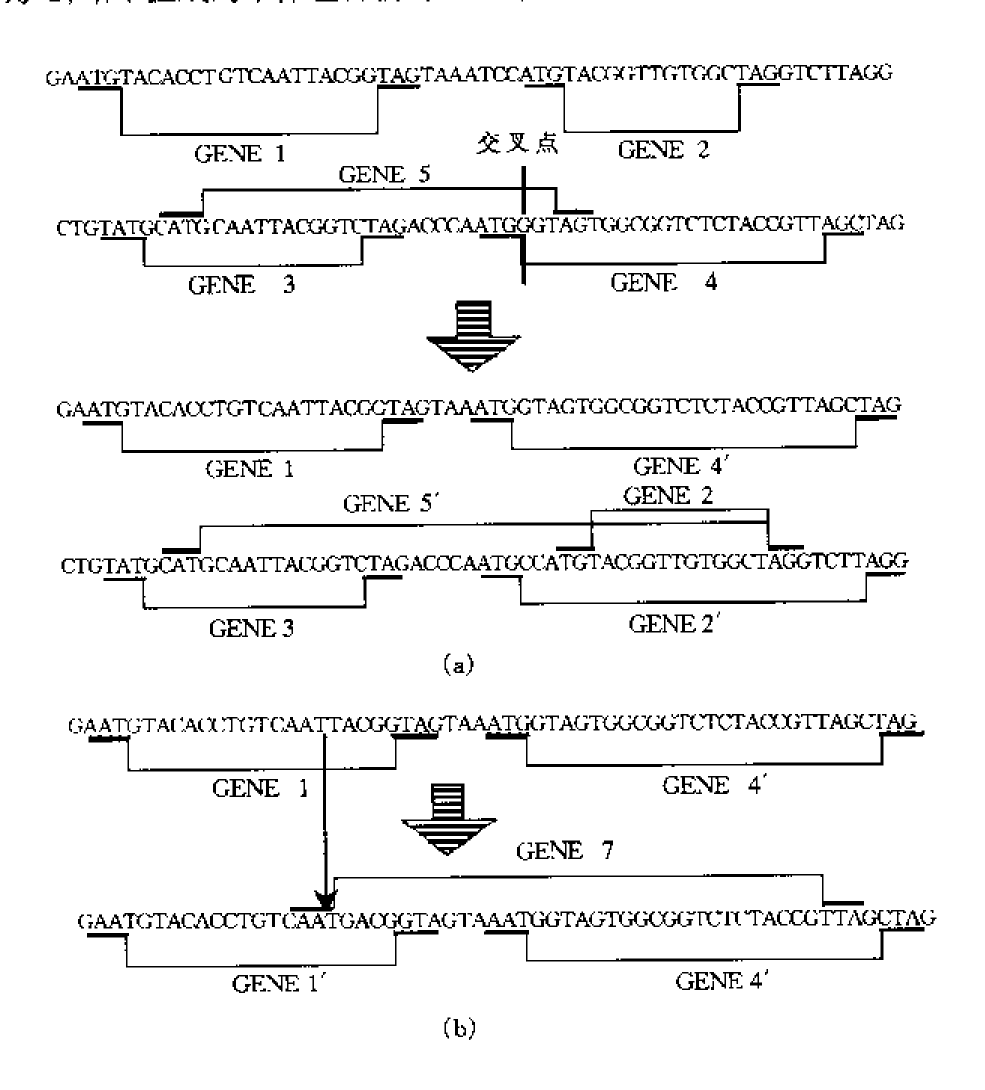


图 9.21 交叉操作与变异操作示意图
(a) 交叉操作； (b) 变异操作

总而言之，DNA 编码方法具有知识表达方式灵活、编码存在冗余和重叠、染色体长度可变、遗传操作设计方便等特点。

2．伪细菌遗传算法(PBGA)

细菌遗传学提供了一个有趣的基因重组机制。细菌能够通过变异将 DNA 物质传递给受体细胞。雄性细胞将基因传送到雌性细胞，雌性细胞接受了雄性细胞的特征。噬菌体(bacteriophage)进入细菌后，就"喧宾夺主"控制并利用细菌的复制、转录和翻译机制，复制噬菌体的 DNA，并以噬菌体的 DNA 来转录 mRNA，再将 mRNA 转译为蛋白质。这样就既有了 DNA，又有了蛋白质，经过组装就产生新的噬菌体。全部过程所需的能、酶、核苷酸、氨基酸等都由细菌供给。噬菌体组装后，细胞膜溶解，噬菌体逸出，再侵入新的细菌。从噬菌体侵入细菌到新的

噬菌体逸出、侵入新的细菌，这一周期称为溶菌周期(lytic period)。溶菌周期一般很短。带细菌基因的噬菌体在侵入新细胞时，就把原来寄主细菌的基因带到新寄主中去了，这个过程被称为转导(transduction)，通过转导单细菌的特征可以传播到整个细胞群体。正是由于这样的遗传重组机制，微细菌进化过程非常迅速。

借助于细菌基因重组机制，发展起来的遗传算法，称为伪细菌遗传算法。首先根据一个细菌的染色体复制多个细菌，对每个新生细菌的相同基因片段进行变异，选择其中细菌的最佳基因片段进行转导，传递给其他的细菌。图 9.22 给出了一个简单的伪细菌遗传操作例子。图中 GENE1 被复制成四个克隆体，除了一个克隆体外，其他三个克隆体接受变异。然后只选择变异操作后四个基因片段中最佳者返回替代原基因片段。对整个细菌群体逐一执行这样的遗传操作之后，再进行常规的选择、交叉和变异操作。

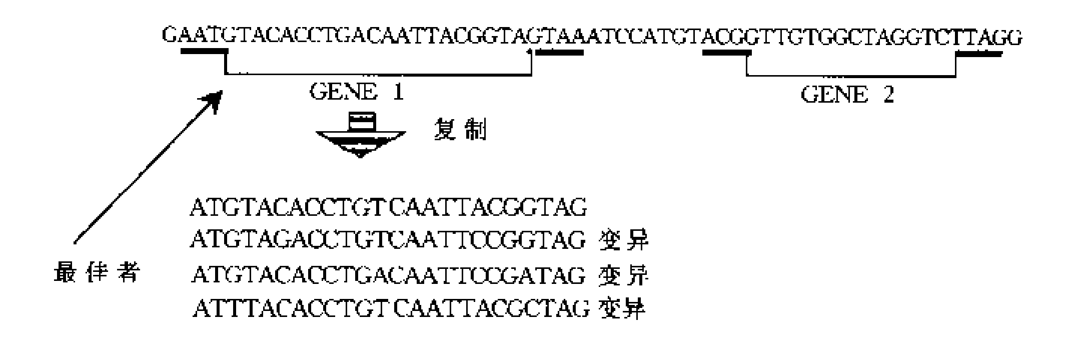


图 9.22　伪细菌遗传操作示意图

3. 应用实例

市桥等是在一个自治控制的机器人追踪试验模拟环境中对上述方法进行实际研究的。如图 9.23 所示，他们设计了两个机器人，在 2.33 m×3 m 室内进行一种追逐游戏实验。追踪机

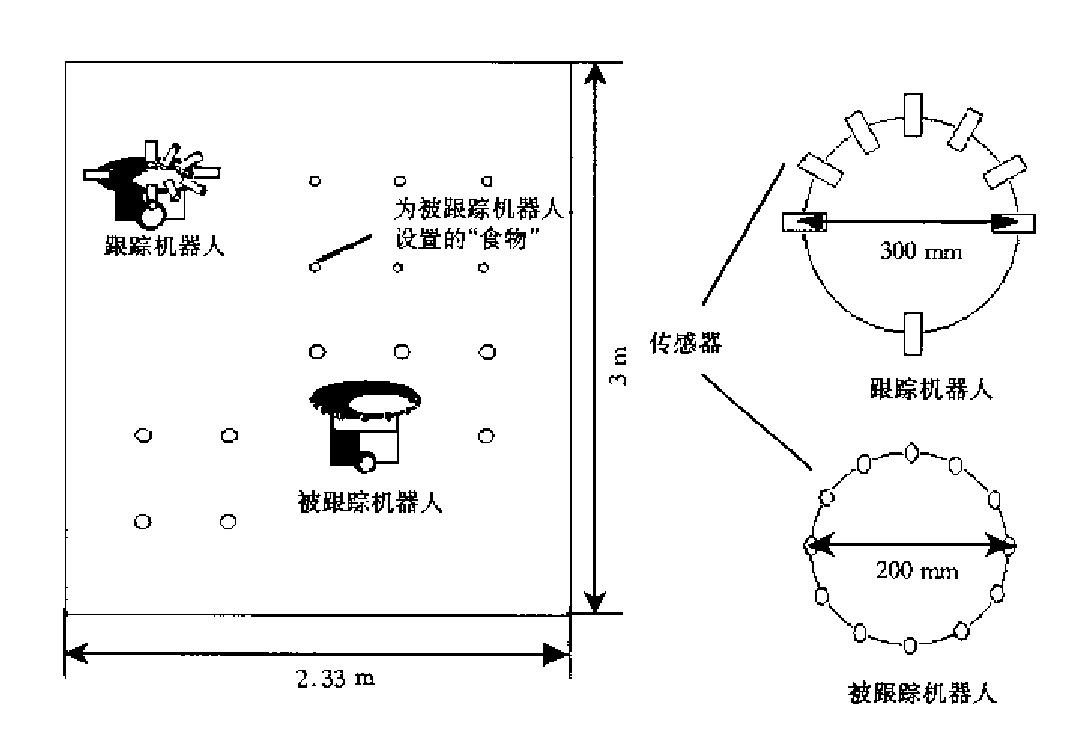


图 9.23　机器人追踪的模拟环境示意图

器人和被追踪机器人的运动半径分别为 150 mm 和 100 mm，追踪机器人装配了 8 个测距超声波传感器，测距范围为 200～1 700 mm，被追踪机器人装配了 12 个红外线传感器，测距范围小于 350 mm。这样，被追踪机器人只有在追踪机器人接近它时才能感知得到。作为追踪机器人需要获知如何赶上追踪对象的行为规则，而被追踪机器人需要获知如何躲过追踪者的行为规则。这些行为规则描述为模糊规则，基于优化的模糊规则控制的机器人在追逐游戏中可以自主地决定行进方向、速度以及远离墙壁等行为。

将机器人的输入变量状态用传感器探测值 D 和当前行进速度 V 描述，输出变量用转角 u 和行进速度 V 描述。机器人的控制规则为 IF－THEN 形式，一个规则集用 DNA 编码方法表示为一个染色体编码。变量的模糊状态用隶属函数表达，染色体中用隶属函数的中央值 Xc 和宽度 σ 代表一个模糊状态。在生物 DNA 中，一个基因的解读开始于密码子 ATG，结束于密码子 TAA，TAG 或 TGA。这里对于一条 IF－THEN 规则的解读，也是始于密码子 ATG，即 ATG 代表 IF。但终止密码子位置不确定。一条规则由一个基因片段翻译而来，64 种密码子对应 20 种氨基酸，每种氨基酸根据其在氨基酸序列中的位置确定具体的含义，这种翻译上的对应关系表参照表 9.3 所示。如图 9.24 所示，在 DNA 染色体中从头部找到起始密码子 ATG，表示开始翻译一条模糊规则。下一个密码子是 GCT，对应丙氨酸 Ala，Ala 表示为输入变量是传感器 Sensor，这样氨基酸序列 Ala，Ser 和 Leu 表示该传感器为第 0 号传感器，即输入变量为 D0；接下来部分 Gly，Cys 确定输入变量 D0 的隶属函数变量 Xc 和 σ；再其次的密码子为 GCC，它也对应于 Ala，但这时 Ala 的含义为 AND，可见密码子位置不同，含义也不同。图 9.25 所示为读解 DNA 染色体中一条模糊规则的流程。当解读完一条规则后，再依次继续解读另一个基因片段获得另一条规则，直到染色体中所有基因片段都解读完，即获得一个规则集。

表 9.3　DNA 染色体中密码子与模糊规则组成成分解码对应关系表

	Phe	Leu	Ile	Val	Ser	Pro	Thr	Ala	Tyr	His	Gln	Asn	Lys	Asp	Glu	Cys	Trp	Arg	Gly
(1) 输入变量	Sensor																	V	
(2) 传感器数	1								2						3				4
(3) 传感器序号	0		9		3		10	2	11		1		8	4	7	5	6		
(4) Xc	Left ←→ Right																		
(5) σ	Narrow ←→ Wide																		
(6) AND/THEN	AND								THEN										
(7) 输出变量	u								V							u and V			

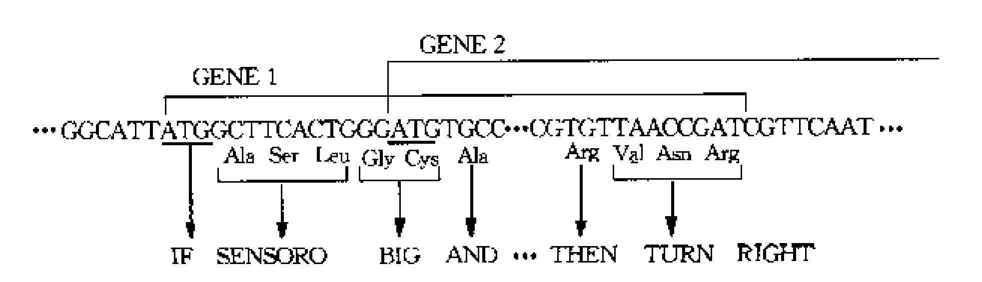


图 9.24　DNA 染色体中模糊规则翻译示意图

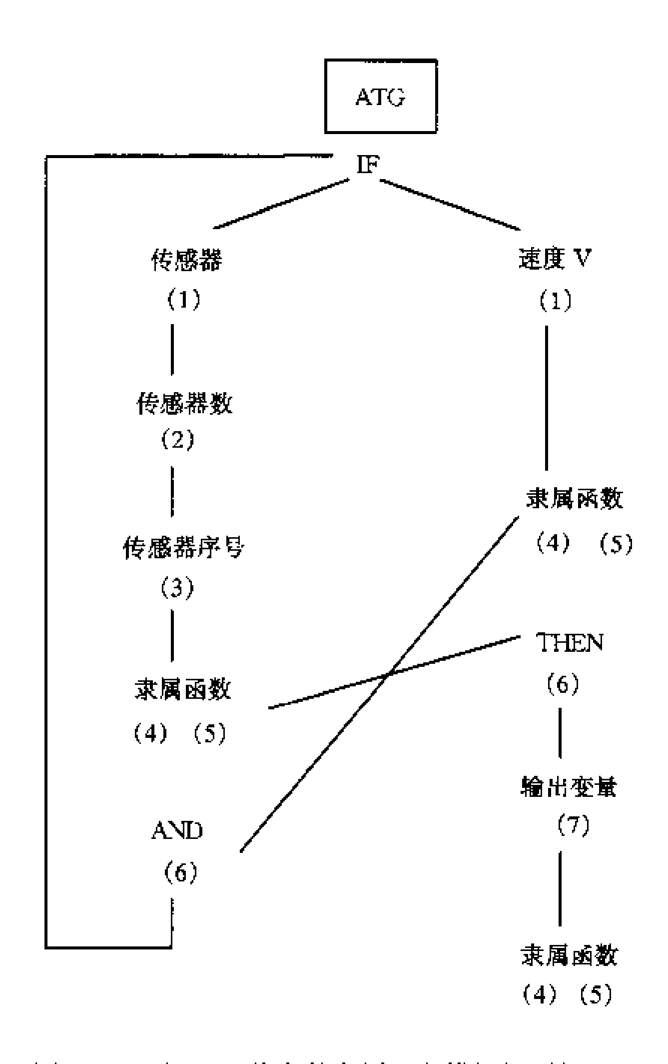


图 9.25　从 DNA 染色体翻译一条模糊规则的流程

在机器人追逐游戏模拟中，机器人个体染色体 DNA 的编码长度为 500，适应度的评价用机器人的环境增益值表示，若机器人碰到墙壁时增益值减少 $E_w = 20$，在固定时间片内（如 30 s 内）追踪方追赶上被追踪方，追踪方的增益值增加 $E_c = 50$，同时被追踪方的增益值减少 $E_a = 20$。若追踪方在该时间片内不能追赶上被追踪方，其增益值减少 $E_m = 20$，同时被追踪方获得一个食物增益值增加 $E_f = 5$。细菌遗传操作的基因片段数为 1，克隆数为 6，变异概率 $P_m = 0.05$。两个机器人的增益初始值 $E_0 = 0$，模拟开始时每个染色体或用于追踪或用于被追踪，随机地从 7 个对手中选择一个作为追踪伙伴，当两者都碰到墙壁或追逐方赶上被追逐方，或达到规定时间片长度时，一次模拟测试结束。每个个体代表的机器人与随机选择的追踪伙伴测试 10 次，计算平均增益值。在完成细菌遗传操作和个体评价之后，将 7 个个体中增益值最小者舍弃掉，在剩余 6 个个体中随机地选择 2 个个体进行常规的交叉和变异操作。在产生的两个新个体中随机选择任意一个作为新一代的机器人个体，并将它的增益值恢复到 0，进入下一代的模拟中。

对于追逐机器人和被追逐机器人个体，轮流进行常规的遗传操作，轮流代数为 100 代。通过 3 000 代以上的模拟实算，可以获得追逐两方机器人的运动控制模糊规则。图 9.26(a)为实算结果中追踪机器人在 3 000～5 000 代计算的平均增益值进化曲线，图中实线部分对应于追踪方使用 DNA 编码方法，而被追踪方使用常规编码方法的结果；虚线部分采用相反的模拟方

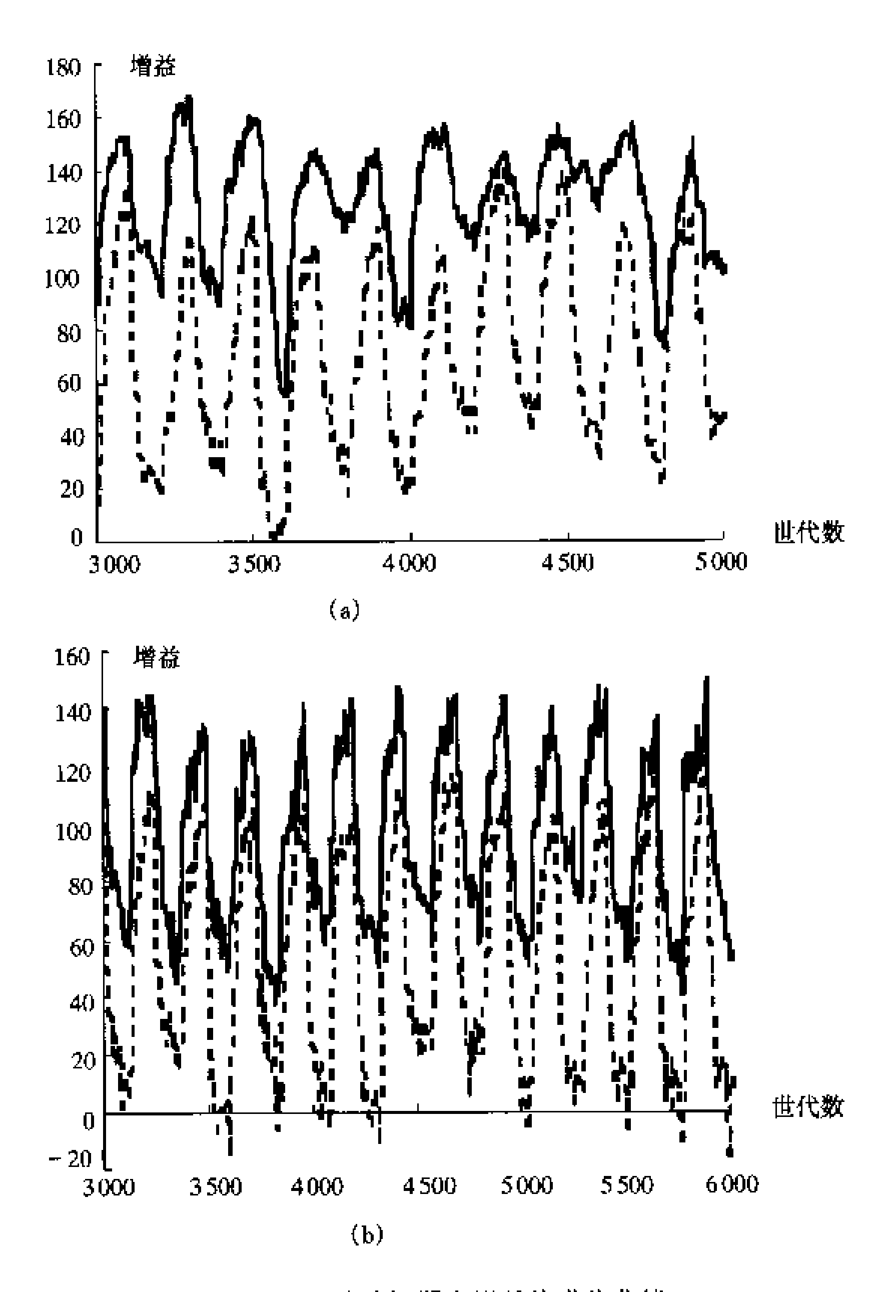


图 9.26 追踪机器人增益值进化曲线

(a) 采用DNA编码与常规编码模拟结果比较;

(b)采用细菌遗传算法和不采用细菌遗传算法的模拟结果比较(均使用DNA编码)

案,追踪方使用常规编码方法,而被追踪方使用DNA编码方法。结果表明DNA编码方法的实效非常明显。此外,由于常规遗传操作的轮流周期性,体现了"道高一尺,魔高一丈"的生存竞争,追踪方的增益值曲线呈波状起伏。图9.26(b)为实算结果中追踪机器人在3000~6000代计算的平均增益值进化曲线,图中实线部分对应于追踪方使用DNA编码和细菌遗传算法,而被追踪方只使用DNA编码方法的结果;虚线部分采用相反的模拟方案,追踪方使用DNA编码,而被追踪方使用DNA编码和细菌遗传算法。结果表明,细菌遗传算法的局部改善机制起了很大作用。

以上介绍基于DNA编码和伪细菌遗传算法相结合的模糊控制规则学习方法,实际上可以看作DNA计算和软计算的一种集成,遗传算法用于模拟生命的自适应和进化,被用于设计

的机器和编制自动化的计算机程序。由于常规遗传算法用于实际问题，尤其是处理复杂的、混淆的和多任务处理时不够灵活，且计算速度慢。基于 DNA 机理的改进的遗传算法，如带双串的 DNA 的遗传算法用于促进 DNA 复制的非对换变异；类似 DNA 编码的"系统描述"用于各种进化系统细胞特定化的模型。在细胞特定化的过程中，每个细胞具有相同的 DNA。硬件建模和计算机仿真显示了细胞特定化具有自组织特性。用生物学 DNA 的发展机理可实现知识的灵活表达，DNA 转录、病毒 DNA 和酶操作也可用这种 DNA 编码方法来实现。

9.5　遗传优化神经网络控制

神经网络是模拟人类生理上的神经机制的计算模型。近 15 年来，神经网络技术已渗透到各个领域，在智能控制、模式识别、计算机视觉、非线性优化、连续语音识别、信号处理等方面取得了巨大的成功和进展，但仍然存在一些难以解决的问题。由于遗传算法能够收敛到全局最优解，而且遗传算法的鲁棒性强，将遗传算法与前馈网络结合起来是很有意义的，不仅能发挥神经网络的泛化的映射能力，而且，使神经网络具有很快的收敛性以及较强的学习能力。遗传算法与神经网络结合主要有两种方式：一是用于网络训练，即学习网络各层之间的连接权值；二是学习网络的拓扑结构。如果考虑具体的优化内容和策略，又可以细分为以下四种方式：

(1) **用于神经网络的训练**　列出神经网络中所有可能存在的神经元，将这些神经元所有可能存在的连接权值编码成二进制码串或实数码串表示的个体，随机地生成这些码串的群体，进行常规的遗传算法优化计算。将码串解码构成神经网络，计算所有训练样本通过此神经网络产生的平均误差可以确定每个个体的适应度。这种方法简单明了，但是用于遗传优化计算的运算量较大，当优化设计解决复杂问题的大规模神经网络时，随着神经元数目的大量增加，连接权值的总数也急剧增加，从而造成遗传算法的搜索空间急剧增大。

(2) **优化神经网络的结构和学习规则等**　利用遗传算法优化设计的不仅是神经网络的结构，而且包括神经网络的学习规则和与之关联的参数。这类方法中有的还利用遗传算法优化设计个体适应度的计算方程。这类方法并不将连接权值编码成码串，而是将未经训练的神经网络的结构模式和学习规则编码成码串表示的个体，因此，遗传算法搜索的空间相对较小。相对于第一种方法，它的缺点是，对于每个选择的个体都必须解码成未经训练的神经网络，再对此神经网络进行传统的训练以确定神经网络的连接权值。

(3) **优化径向基神经网络**　介于对所有连接权值都编码的第一种方法和不对任何连接权值编码的第二种方法之间，这种方法构造一空间填充曲线(space - filling curve)来确定神经网络输入层到第一层的函数，它的搜索空间与计算量都是适中的，但只能用于基于径向基神经网络(RBF network)的优化设计。

(4) **同时优化神经网络的结构和连接权值**　即同时对神经网络的结构和连接权值都进行编码。Vittorio 还提出了粒度(granularity)编码方法以提高连接权值的优化精度。Vittorio 的粒度控制一方面加快了收敛到一定优化精度的搜索时间，但另一方面会引起个体适应度的剧烈不连续变化，从而又间接地导致遗传算法收敛速度变慢。

在 8.4 节中，我们已经介绍了协同进化遗传算法用于神经网络训练的方法。本章将结合一个实例，介绍这一方法在过程控制中的应用。然后将介绍遗传算法应用于神经网络结构优化设计的一般方法。最后，作为这方面的最新发展，介绍遗传程序设计应用于神经网络学习规

则发现方面的研究。

9.5.1 协同进化的过程控制

在前面已经介绍了协同进化遗传算法(CGA)以及一个应用于分类任务的神经网络训练实例。这里,将CGA应用于一个过程控制测试问题(benchmark problem)——生物反应器过程的神经网络控制。如图9.27所示,一个生物反应器为一个盛有水、培养液和生物细胞的水槽。培养液和细胞按流速 r 引入水槽,含培养细胞的流出液流速与培养液的流入流速相等,使槽内液体量保持恒定。该过程的过程参数为细胞数量和培养液量,控制目标使槽内细胞数量达到并保持预定的数量水平。

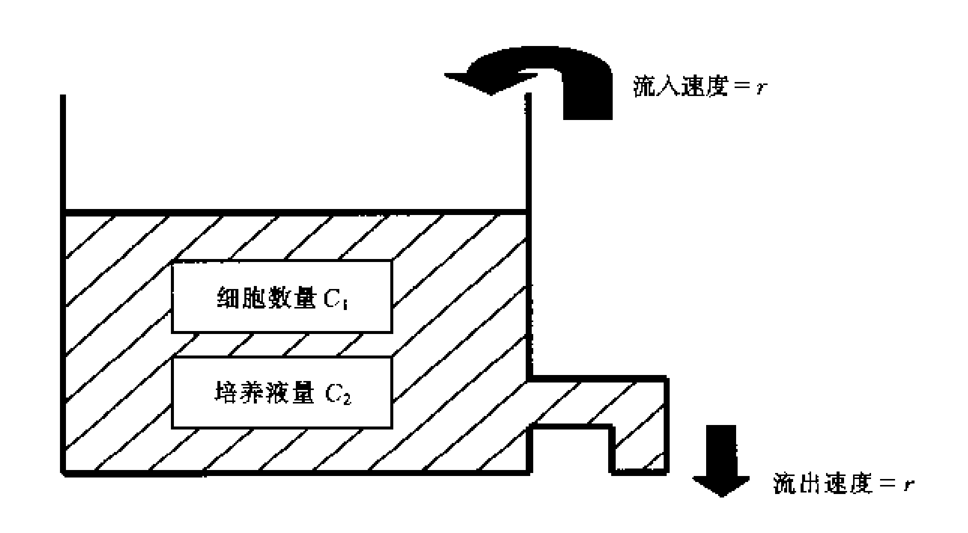


图9.27 生物反应器示意图

生物反应器过程控制虽然控制参数较少,比较容易模拟实现,但它的非线性特性造成了控制的困难。因为控制参数很小的变化,就可能使过程处于不稳定。假设生物反应器的状态变量分别为细胞数量 C_1、培养液量 C_2 和流速 r。变量 C_1 和 C_2 经一定的变换转化为区间[0,1]的值,r 转化为[0,2]区间值。系统的状态可由($C_1[t]$, $C_2[t]$, $r[t]$)表示,初始状态为($C_1[0]$, $C_2[0]$, $r[0]$)。系统动态可用下面两个差分方程描述:

$$C_1[t+1]=C_1[t]+\Delta\cdot(-C_1[t]\cdot r[t]+C_1[t]\cdot(1-C_2[t])\cdot e^{\frac{C_2[t]}{\gamma}}) \tag{9.37}$$

$$C_2[t+1]=C_2[t]+\Delta\cdot(-C_2[t]\cdot r[t]+C_2[t]\cdot(1-C_2[t])\cdot e^{\frac{C_2[t]}{\gamma}})\cdot\frac{1+\beta}{1+\beta-C[t]} \tag{9.38}$$

上式中,β 为细胞生长参数($\beta=0.02$),γ 为培养抑制参数($\gamma=0.48$),由这两个参数分别控制细胞的生长速度和培养液消耗。Δ 为采样间隔($\Delta=0.01$ s)。上式表示了反映器在 Δ 之后的状态。

假定模拟时间为50 s,控制变量 r 每隔0.5 s调节一次。控制器的输入变量离散化为 $C_1[t]$和 $C_2[t]$,输出变量离散化为 $r[t]$, $t=50, 100, \cdots, 5\,000$,在离散间隔内 $r[t]=r[t-1]$。控制过程的目标函数可以表示为下列累积误差形式:

$$\sum_{\in\{50,100,\cdots,5\,000\}}(C_1[t]-C_1^*[t])^2 \tag{9.39}$$

这里，$C_1^*[t]$为时间 t 的预定细胞数量水平。

图9.28所示为以上过程的神经控制器结构，为一个标准的多层感知器，含有一层12个结点的隐层，输入层有两个结点，对应于 $C_1[t]$和 $C_2[t]$，输出层有一个结点，对应于 $r[t]/2$。协同进化遗传算法的第一个种群为神经网络的解种群，解个体的染色体编码为所有连接权值的编码，初始解个体的所有权值取值范围为$[-1,1]$。第二个种群有200个个体，每个个体由状态变量三元组$(C_1[t],C_2[t],r[t])$构成个体的编码，其中 $C_1[t]\in[0.9C_1^*,1.1C_1^*]$，$C_2[t]\in[0.9C_2^*,1.1C_2^*]$，$r[t]\in[0.9_r^*,1.1_r^*]$内的均匀随机数。因此这个种群可称为"过程状态种群"，种群中的个体称为"过程状态个体"，这两个概念与协同进化遗传算法中介绍的"测试种群"、"测试个体"含义是一样的，只是因具体问题不同改变了称谓而已。

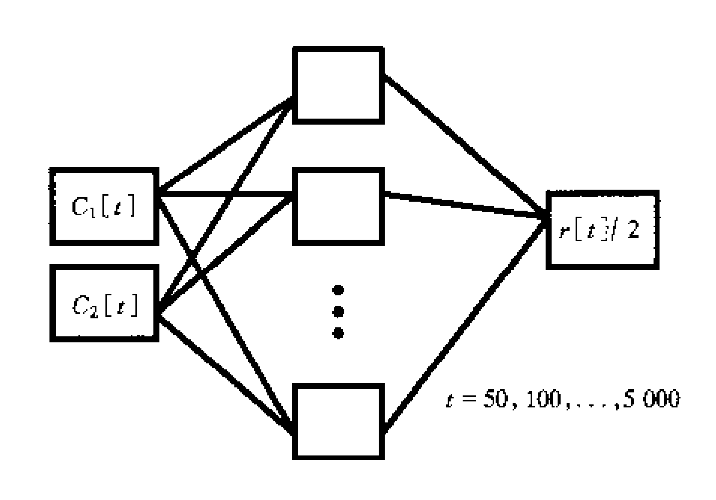


图9.28　生物反应器过程的神经网络控制器结构

协同进化遗传算法中，种群的交互作用通过代表一个神经网络的解个体和过程状态个体之间的遭遇来实现。当发生一次遭遇时，将状态个体作为初始状态$(C_1[0],C_2[0],r[0])$代入系统动态方程(9.37)和(9.38)，通过50次迭代运算获得 $C_1[50]$，$C_2[50]$，作为神经网络个体的输入，按标准的前馈运算获得输出结果，输出结果的二倍即为 $r[50]$，这50次迭代运算称为一个循环。经过如此100次循环运算，按式(9.39)计算过程的累积误差，并将该误差项作为过程状态个体增益评价值，而遭遇的另一方——神经网络解个体的增益值正好取该误差项的负值。将最近的20次遭遇的增益均值作为个体的适应度评价，这里，较好的神经网络个体能够获得较小的增益均值(负值)，而较好的过程状态个体能够获得较大的增益均值(正值)。如果从一个过程状态个体表示的初始状态开始，神经网络个体对应的控制器不能获得很好的控制性能，这个状态个体则被视为一个好个体。

对于上述生物反应器控制，假定目标状态$[C_1^*,C_2^*,r^*]=[0.120\,7,0.880\,1,0.75]$，对应于一个稳定状态。考察基于CGA训练神经网络控制器的过程模拟结果。

应用CGA训练的最佳神经网络控制器控制生物反应过程，模拟时间取50 s，获得系统过程动态。如图9.29所示，从细胞数量 C_1 的响应曲线分析，系统动态特性很好，经过快速的衰减，稳定在0.121 44，稳态误差只有0.6%。

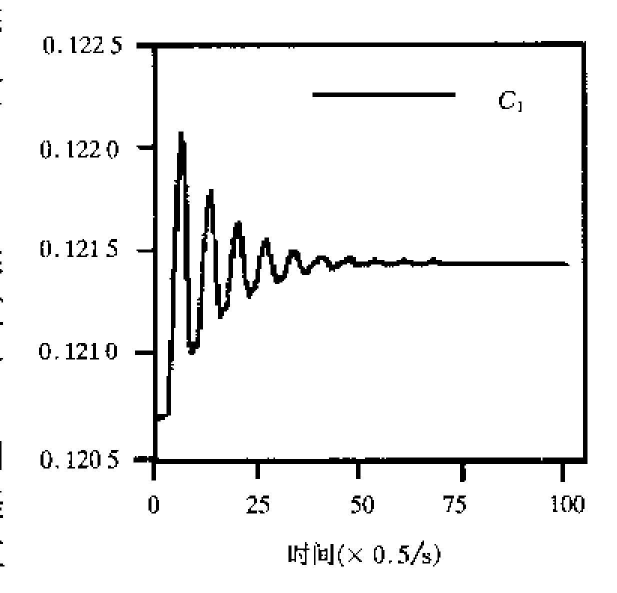


图9.29　经CGA训练获得的最佳神经控制器控制的生物反应器过程曲线

训练获得的最佳神经控制器过程曲线,其中 $C_1-\text{opt}$ 对应于最佳初始状态 $[C_1^*, C_2^*, r^*]$ 的响应曲线,C_1-ex1 和 C_1-ex2 对应于过程状态种群中适应度最大的两个个体所代表的初始状态的响应曲线。作为比较,图 9.30(b)中 $C_1-ex199$ 和 $C_1-ex200$ 对应于过程状态种群中适应度最小的两个个体所代表的初始状态的响应曲线。不难发现,从适应度小的状态个体比适应度大的状态个体在与最佳解个体"遭遇"时,系统的动态性能要好,即累积误差小。图中,响应曲线 $C_1-ex199$, $C_1-ex200$ 比 C_1-ex1 和 C_1-ex2 的震荡幅度要小得多。这体现了来自于两个种群的两个个体在遭遇竞争时此消彼长的一种平衡。

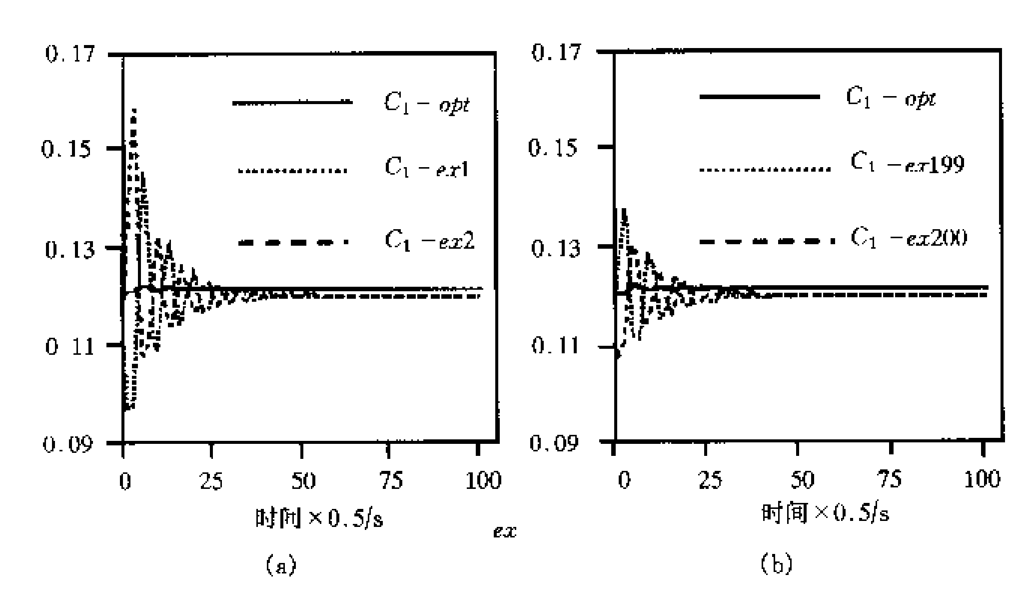


图 9.30 最佳神经控制器在不同初始状态下模拟获得的过程曲线

(a) 高适应度的初始状态; (b) 低适应度的初始状态

在模拟过程中,过程状态种群中的个体对应的初始状态($C_1[0]$, $C_2[0]$, $r[0]$)与目标状态 $[C_1^*, C_2^*, r^*]$ 的三维笛卡儿空间距离,称为个体的稳态距离。分析历代个体的稳态距离与其在过程状态种群中的序位关系,可以发现种群演化的规律。图 9.31(a)和(b)分别给出了初代

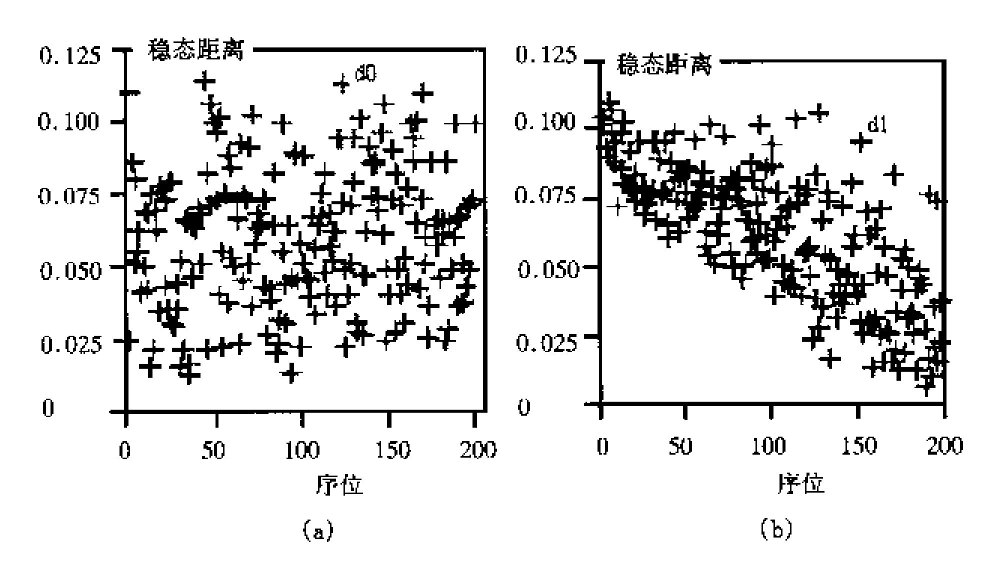


图 9.31 过程状态种群中个体稳态距离与序位之间的关系

(a) 初始代; (b) 末代

和进化末代的情形。在初代个体的稳态距离呈随机分布,在进化末代,最佳的过程状态个体远离目标状态,而最差的过程状态个体显然易于控制器控制,距离目标状态很近。

对过程控制而言,控制的目标是要求满足这样的控制准则,即将一个或多个控制变量保持在预定的控制水平。上面给出的生物反应过程控制应用协同进化遗传算法,很好地适合了这一要求。通过引入表现过程状态的另一个种群,与控制器解种群通过适应度交互,完成一个基于生态竞争进化的过程。两个种群适应度评价采用 LTFE 方法,促成了遗传学习与后天学习的结合。从优化搜索的角度而言,实际上可视为全局搜索与局部搜索的一种有机的结合。这种方法经适当的扩展,可以应用到其他类型的控制问题,例如,对于含有多种噪声模式的控制对象,可以将不同噪声模式构成一个种群,将控制器的测试集作为另一个种群。此外,引入共生关系的 CGA,对于提高模拟计算性能和控制性能很有意义。

9.5.2 神经网络结构的优化设计

神经网络拓扑结构的设计是神经网络设计的重要内容,由于没有足够的理论指导,这方面更多地依靠有经验的专家通过摸索来确定。前馈神经网络的研究较多,其结构相对简单,并且有 BP 学习算法或其他学习算法,其结构确定的关键问题是隐结点层数和个数的确定,这方面已有许多研究,并且相对比较简单。对于具有反馈结构的神经网络,目前尚缺乏有效的学习算法,而且结构确定很困难。这里以一个异或(XOR)问题的神经网络训练为例,讨论一种同时确定神经网络结构和权值的遗传优化方法,能够处理各结点任意连接的神经网络,对结点之间的连接方式不作严格的要求。

为简化运算,对神经网络算法作如下限制:

① 各神经元的连接权值只在 0, 1, −1 之间取值,当两个神经元之间的连接权值为 0 时,说明它们之间没有联系;

② 隐层结点和输出结点的输出值不反馈给输出结点,隐层结点与输出结点之间可以反馈;

③ 神经元状态的迁移是异步进行的。设定神经元 $u_i(i=1,2,3,\cdots)$ 在 t 时刻的状态 $O_i(t)$ 时,其值为 0 或 1。设 $u_{ji}(j=1,2,\cdots,N_i)$ 是与神经元 u_i 相连的神经元,它在时刻 t 的状态为 $O_{ji}(j=1,2,\cdots,N_i)$ 时,从 u_{ji} 到 u_i 的连接权值为 w_{ji},则 u_i 在时刻 $t+1$ 的状态为:

$$O_i(t+1)=\begin{cases}1, & \sum_{j=1}^{N_i} w_{ji}\cdot O_{ji}(t)>0\\ 0, & \sum_{j=1}^{N_i} w_{ji}\cdot O_{ji}(t)\leqslant 0\end{cases}\tag{9.40}$$

这就是神经元状态迁移的规则。

在应用遗传算法优化神经网络结构时,我们考虑以下几个问题:

1. 染色体编码

神经网络的结构如图 9.32 所示,每个神经网络结构需要表示为一个遗传算法个体染色体编码,才可以用遗传算法进行优化。

设神经网络有 N 个神经元,序号是从 1 到 N 排列的输入层、隐层、输出层结点。设计一个 $N\times N$ 矩阵,表示其互连结构。如表 9.4 所示,在矩阵中(i,j)的元素表示从第 i 个神经元到第 j 个神经元的连接关系,“0”表示没有连接,“±1”表示其连接权值为 ±1,“x”表示没有关

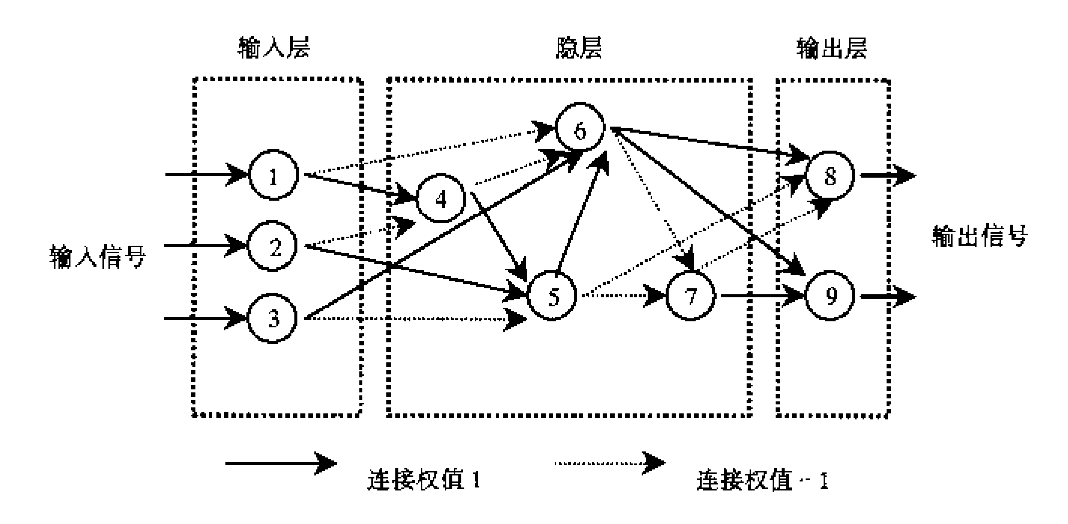


图 9.32 神经网络结构示意图

联。因此，图 9.32 所示的结构表示为与表 9.4 等价的矩阵形式。

设神经网络有 N_{input}, N_{hidden}, N_{out} 个神经元，总数为 $N = N_{input} + N_{hidden} + N_{output}$。由于"*"是不包含在优化变量中，所以该问题的搜索空间为：

$$N_{state} = 3^{N\times(N_{hidden}+N_{output})} \tag{9.41}$$

表 9.4 图 9.32 所示神经网络中神经元连接关系表

to \ from	1	2	3	4	5	6	7	8	9
1	x	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x	x
3	x	x	x	x	x	x	x	x	x
4	1	-1	0	x	x	x	x	x	x
5	0	1	-1	1	x	x	x	x	x
6	-1	0	1	-1	1	x	x	x	x
7	0	0	0	0	1	-1	x	x	x
8	0	0	0	0	-1	1	-1	x	x
9	0	0	0	0	0	1	1	x	x

由表 9.4 所示的神经元连接关系，可以将该神经网络对应的遗传编码表示为 0, 1, -1 组成的数字串形式，将元素(4, 1)到元素(9, 7)自左到右、自上而下顺序连接起来，组成如下的染色体编码：

1 -1 0 ¦ 0 1 -1 1 ¦ -1 0 1 -1 1 ¦ 0 0 0 0 1 -1 ¦ 0 0 0 0 -1 1 -1 ¦ 0 0 0 0 0 1 1

2. 适应度定义

对于神经网络个体适应度的定义，设神经网络输入 n_{total} 次信号，神经网络输出 $n_{correct}$ 次正确解，则适应度函数可以采用以下形式：

$$f = \frac{n_{correct}}{n_{total}} \tag{9.42}$$

适应度函数值在[0, 1]区间内，越接近于 1 的个体，其输出信号正确率越高。

3.遗传操作的设定

如图9.33所示,为生成子代种群的一种方法。首先将当代种群的个体按适应度由大到小进行排序,然后选择一定比例的下位个体淘汰掉,淘汰比例一般为40%。在上位个体中实行均匀交叉,生成的子个体填补到种群中,以保持种群规模不变。最后,实行变异操作(变异概率为0.01),生成子代种群。

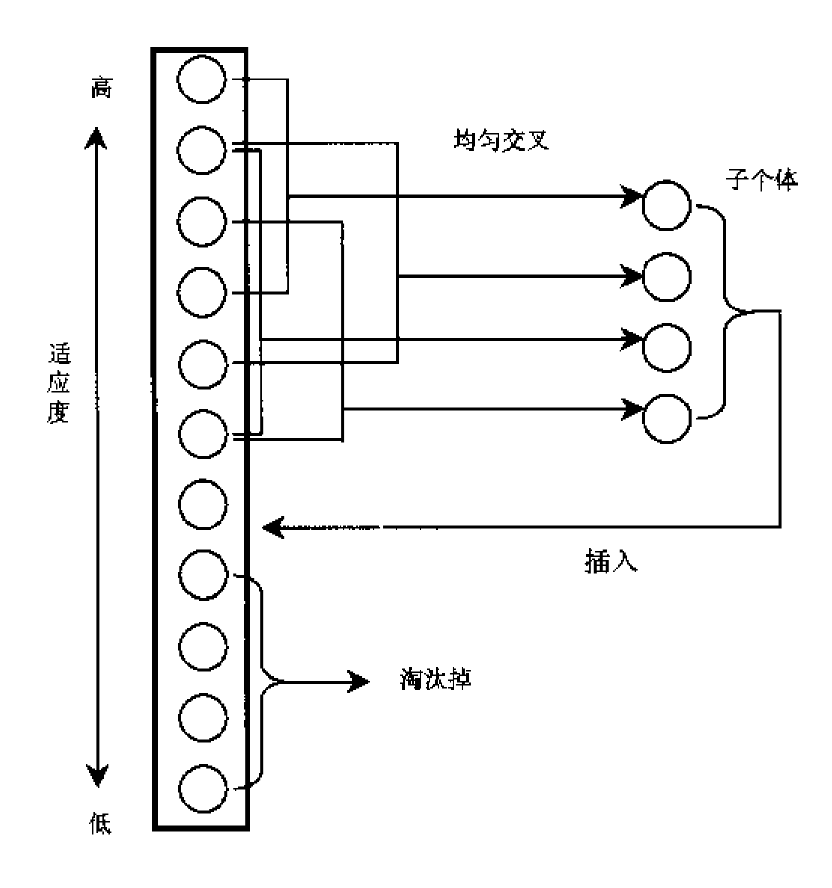


图9.33　子代个体生成方法

根据上述编码方法、适应度定义以及遗传操作的设定,我们以描述异或问题的神经网络结构的优化设计为例,进行了模拟实算。设定网络有2个输入结点和1个输出结点,选取6个隐结点(一般为5~10个),异或问题的输入输出信号对:

$$(0,0)\rightarrow 0$$
$$(0,1)\rightarrow 1$$
$$(1,0)\rightarrow 1$$
$$(1,1)\rightarrow 0$$

图9.34为XOR神经网络优化模拟的结果。图中左上部分为异或问题4个训练样本,左下部分为历代种群中最大适应度个体对应的神经元连接矩阵,其中"+"表示连接权值为1,"-"表示连接权值为-1,"0"表示没有连接,"x"表示没有关联。右上部分为遗传算法中历代适应度演化曲线,作为与遗传算法比较,右上部分显示出一般随机搜索方法历代"适应度"变化情况。可见,遗传算法方法只需要25代左右就可以获得最大适应度的个体。图9.35所示的是末代种群中最优个体解码后获得的最佳神经网络结构和连接权值。将该网络结构进行简约处理,可得到图9.36所示的结构。简约处理的原则有两个:若结点无论输入什么信号,永远输

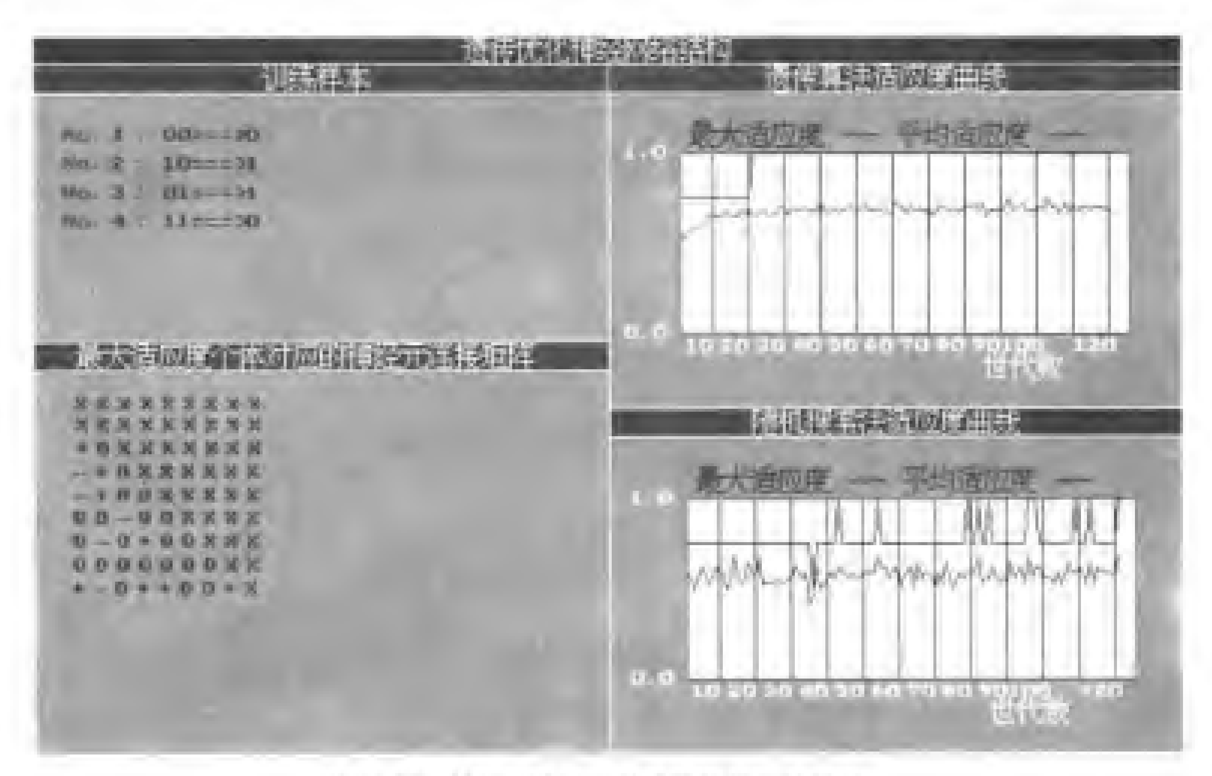


图 9.34 遗传优化异或神经网络结构的模拟

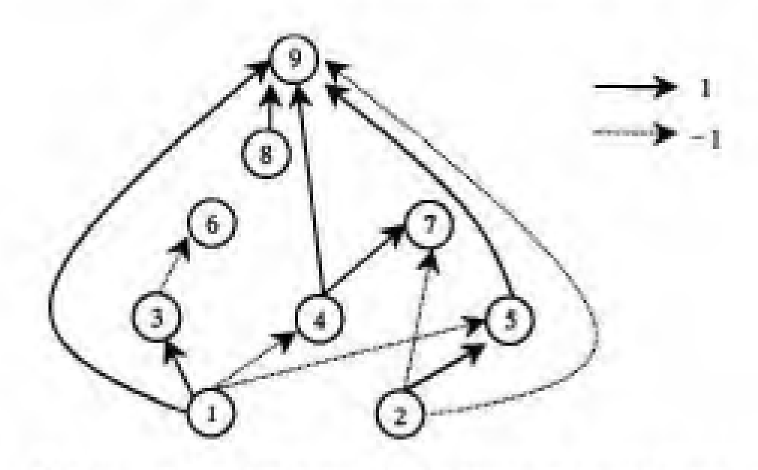


图 9.35 遗传算法优化获得的最佳神经网络结构和连接权值

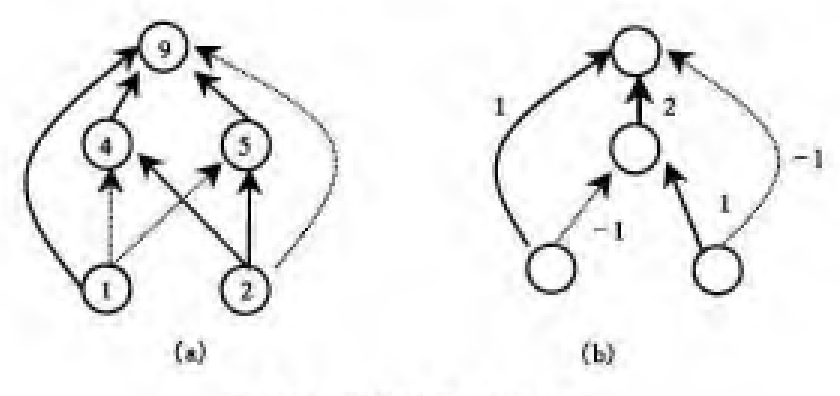


图 9.36 优化结果简约化处理

(a)简约化的结构； (b)等价的结构

出为 0，则该神经元可以从网络中去掉；另外，若结点到输出结点无可达信号通道，则该神经元也可从网络中去掉。依照这两个原则，图 9.35 中的隐结点 3，6，7 和 8 可以去掉，得到图 9.36(a)所示的网络。如果将两个隐结点 4 和 5 合并，又可以进一步简化为图 9.36(b)所示的网络，实际上两者是等价的。

参见附录 II－3 遗传优化神经网络结构源程序。

值得注意的是，遗传优化神经网络结构所涉及的首要问题是如何将网络结构映射到一个染色体编码形式，上例中的编码方式可以归为直接编码方式的一种，当处理大规模神经网络结构设计时，存在染色体长度过长的问题。1990 年，H. Kitano 提出一种矩阵语法生成编码方法，用类似于 L－System 分形结构的图生长规则构造表示网络结构的连接矩阵，而将一些简单的网络生长语法规则编码于染色体中，通过生成规则的进化修改，最后生成满足问题要求的网络结构。

生成规则形式如下：

$$S \rightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} \tag{9.43}$$

规则的前部为一符号，后部为一 2×2 矩阵，其中每个元素属于符号集 $\{A, B, \cdots, Z, a, b, \cdots, p, 0, 1\}$。

一个网络结构编码由变量部分和常量部分组成。编码中 S 表示第一条规则的开始，大写字母 $ABCD\cdots Z$ 表示变量，小写字母 $a, b, \cdots, p$ 为 16 个常量，每个常量也表示为一条规则，例如：

$$a \rightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad b \rightarrow \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad c \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad e \rightarrow \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

若编码中含有数字 0 和 1，则将它们表示为如下形式：

$$0 \rightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad 1 \rightarrow \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

例如，对于一个结构编码 SABCDAcpacBaaaeCaaaaDaaab，依据上述语法规则解码成连接矩阵形式，则有：

$$S \rightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} \rightarrow \begin{pmatrix} c & p & a & a \\ a & c & a & e \\ a & a & a & a \\ a & a & a & b \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{9.44}$$

上式对应的网络结构如图 9.37 所示。

这种编码方法不仅适用于多层前馈网络，且适用于其他网络模型（反馈网络除外）。虽然这种方法尚有一些缺陷，但是在网络结构的自动设计中有很大的发展潜力。

1992 年 F. Gruau 提出了另一种相似的方法，称为细胞编码（cellular encoding），用遗传程

序设计中的树结构描述神经网络结构和权值。有人将这种基于图文法的遗传算法与线性串结构描述的遗传算法区分开来，统称为图文遗传算法(Genetic Algorithms based graph grammar)。

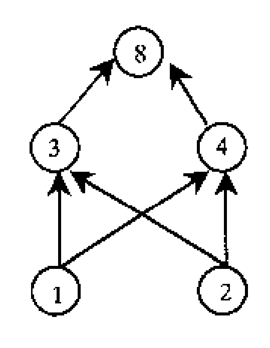


图 9.37 解码结果对应的网络结构

为说明方便，这里根据图文法解释将一个文法树(grammar tree)解码为一个异或神经网络的过程，如图 9.38 所示。文法树类似于 L－System 编码为一个符号树，每个树结点符号都有特定的含义。开始状态在输入指针细胞和输出指针细胞之间设置一个被继承细胞，被继承细胞中包含了文法树的“遗传码”的备份，每个细胞都有一个指向文法树中某结点的指针。

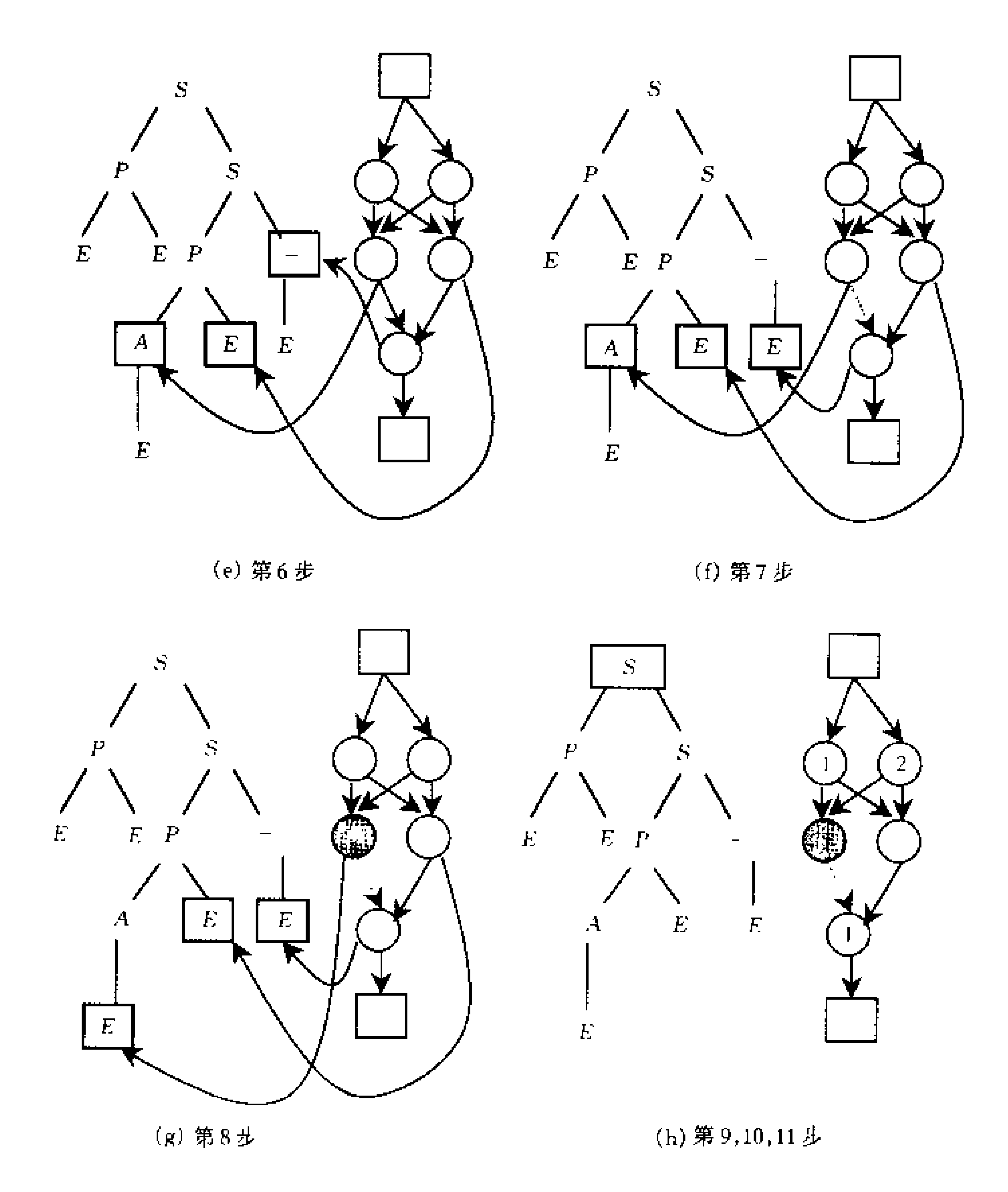
文法树中的符号“S”表示对指向该结点的细胞执行的一次顺序分裂(sequential divide)，顺序分裂的结果产生上、下顺序连接的两个子细胞，上置子细胞继承父细胞的所有输入连接，其指针指向 S 结点的左分支结点；下置细胞继承父细胞的所有输出连接，其指针指向 S 结点的右分支结点。图中第 1 步表示执行一次顺序分裂。

输入指针细胞 被继承细胞 输出指针细胞

(a) 开始状态 (b) 第 1 步

(c) 第 2 步 (d) 第 3,4,5 步

图 9.38 神经网络的细胞编码



(e) 第6步

(f) 第7步

(g) 第8步

(h) 第9,10,11步

续图9.38 神经网络的细胞编码

文法树中的符号"P"表示对指向该结点的细胞执行的一次并列分裂(parallel divide),并列分裂的结果产生左右并列的两个子细胞,它们同时继承父细胞的所有输入连接和输出连接,左置子细胞指针指向P结点的左分支结点;右置子细胞指针指向P结点的右分支结点。图中第2步表示继续执行一次并列分裂。

文法树中的符号"E"表示终止该分支的细胞扩展。符号"A"表示增加隐结点的阈值,"−"表示将指向该结点的输入连接权重设置为−1。

图中第3,4,5步表示继续执行顺序分裂,第6步表示继续执行一次并列分裂,第7步表示再引入一个负权重连接(图中虚线表示),第8步表示再执行一次增加权值操作(图中实心表示的细胞),第9,10,11步为最后处理几个终止结点。

除了上述文法符号外,还有一个未涉及的符号"R",它用于复用神经网络的一部分。当某

细胞指向符号 R 时，指针向根结点方向回退一定步数，再在到达结点位置继续扩展。回退的步数由 R 结点的子结点约定。如果回退步数为 0，则不做回退扩展，而是继续向前扩展。

1995 年 Gruau 将上述图文遗传算法应用于一个 6 足移动机器人神经网络控制器的设计中，获得很好的控制效果。

这种图文法编码方法有三个优点：

① 属于变长度编码方法，相对而言受问题大小的影响小。

② 仅产生合法的神经网络结构，从而避免了大量无效神经网络参与进化过程，加快了进化学习过程。

③ 这种方法直接、简单，易于理解。

但基于二进制编码的模式定理不能简单推广到基于图文法编码的遗传算法，因此这方面的理论研究将是很有意义的工作。

9.5.3 神经网络学习规则的自动设计

当神经网络应用于控制系统时，多采用多层前馈神经网络。多层前馈神经网络的学习算法主要地依靠反向传播法。其实，正是由于该算法的有效性，使隐层结点权值的修正成为可能，从而发展了多层前馈神经网络的应用。

下面我们先简单分析一下标准的反向传播算法(standard backpropagation, SBP)及其改进。

反向传播算法实际上是一种监督学习方法。它利用均方误差和梯度下降法来实现网络连接权值的修正。对网络权值修正的目标是使网络误差平方和最小。算法首先对网络连接权值设置小的初值，然后选择一个训练样本计算相对于该样本的误差梯度。这涉及到两个过程：一个是前向过程，以便计算各神经元的输出直至求出网络的输出；另一个是反向过程，以便逐层计算梯度。一旦梯度已知，便可按照学习规则更新连接权值，直至算法的收敛。

如图 9.39 所示，输入层为第 0 层，输出层为第 k 层。设第 l 层的第 i 个神经元为 u_i^l，w_{ij}^l 为第 l 层的神经元 i 到第 $l+1$ 层的神经元 j 之间的连接权值。并设定输入模式集 $\{x_1, x_2, \cdots, x_m\}$，相应的输出模式集为 $\{t_1, t_2, \cdots, t_m\}$，即存在 m 个训练模式 (x_p, t_p)，$p=1,2,\cdots,m$。

输入训练模式 (x_p, t_p) 时，对输入层神经元 u_i^0 的输出 O_{ip}^0 与 x_i^p 是一致的。对 $l+1$ 层神经元 u_i^{l+1} 而言，可用下式计算其输入。

$$net_{ip}^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l O_{jp}^l - \theta_i^{l+1} \tag{9.45}$$

式中 $O_{jp}^l = f(net_{jp}^l)$ 为 u_j^l 神经元的输出，θ_i^{l+1} 为神经元 u_i^{l+1} 的偏置量，通常为表示方便将它处理为连接权值固定为 1 的一个处理单元。

在输出层上神经元 u_j^k 的输出误差为 ε_{ip}^k：

$$\varepsilon_{jp}^k = t_{jp} - O_{jp}^k \tag{9.46}$$

隐层神经元的输出误差为下一层中与其连接的神经元输出误差的加权和。网络的性能可以用误差平方和进行评价。

$$E = \sum_{p=1}^{m} \sum_{i=1}^{n_k} \varepsilon_{ip}^2 \tag{9.47}$$

连接权值 w_{ij}^l 的更新量采用下列形式：

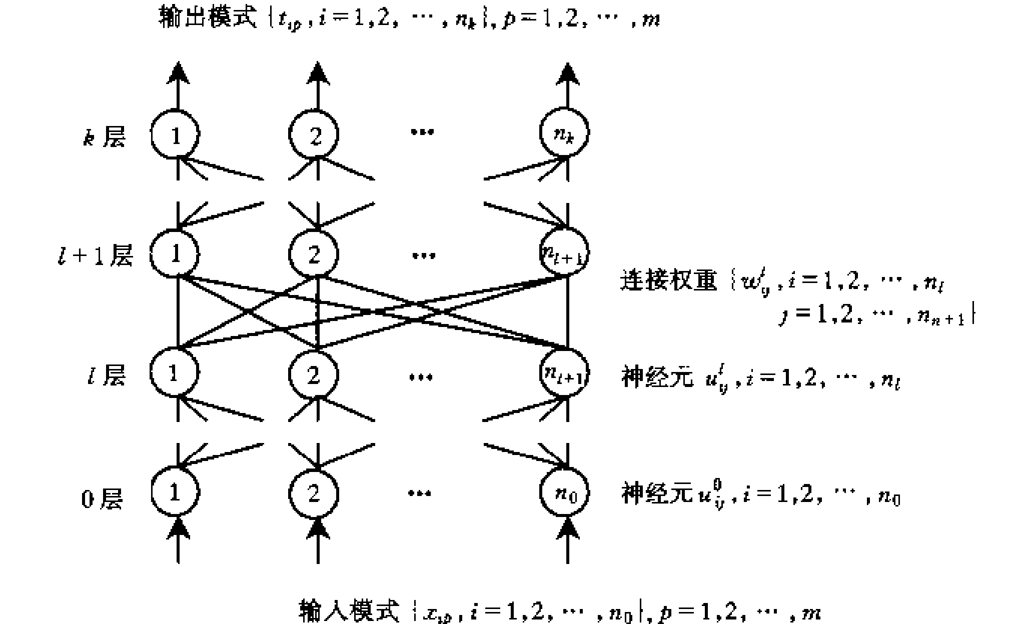


图 9.39　反向传播网络模型

$$\Delta w_{ij}^l = \eta \frac{\partial E}{\partial w_{ij}^l} \tag{9.48}$$

式中 η 为学习速率参数。通常也将该式称为 SBP 学习规则。

BP 学习算法对神经网络进行训练时，由于需要提供教师信号对网络训练，对时变系统这种训练难以达到很高的精度。此外，由于 BP 学习算法是基于梯度下降这一本质，不可避免地带来以下三个缺点：

① 学习过程收敛速度慢；

② 算法不完备，容易陷入局部极小点，而当学习速率设置高时，可能产生振荡；

③ 鲁棒性不好，网络性能对网络的初始设置比较敏感。

为了提高 BP 算法的收敛速度，许多研究者提出了改进方法。其中以采用自适应学习机制的“弹性反向传播法”(resilient backpropagation, rprop)比较有效，虽然这种方法比 SBP 要快得多，但另两个缺点仍然存在。

遗传算法的发展为我们提供了一个全局的、稳健的搜索优化方法，通常固定学习规则表达式形式，专门用遗传算法优化学习规则中的参数。为了搜索更为广阔的学习规则空间，将学习规则的整个表达式作为进化对象。1994 年，Benjio 提出了基于遗传程序设计方法的学习规则发现方法。其通用形式为：

$$\Delta w_{ij}^l = f(w_{ij}^l, x_{jp}^l, t_{ip}, O_{ip}^l) \tag{9.49}$$

式中 f 为需要通过发现的函数。

以 XOR 神经网络为例，使用一个三层前馈网络结构(2 个输入结点，1 个隐结点和 1 个输出结点)，初始连接权值随机地在区间[-1,1]内取值。将隐层学习规则固定为 SBP 学习规则，将输出层学习规则作为搜索对象，每条学习规则根据一个树结构编码为 S-表达式形式，函数集为{+, -, *}，终结点集为{w_{ij}^l, x_{jp}^l, t_{ip}, O_{ip}^l, 1, 2}，树结构的最大深度为 3。学习规则

的适应度定义为：

$$f = \begin{cases} E_{max} - E, & \text{神经网络不学习时} \\ C_{max} - C_{min}, & \text{神经网络学习时} \end{cases} \tag{9.50}$$

式中，C_{min}，C_{max}分别是神经网络采用该学习规则时达到收敛的最小和最大学习次数。E_{max}和C_{min}为预定常数。

遗传程序设计运行参数设置为：种群规模 100，运行代数 500，交叉概率 0.9，变异概率 0.01。经过 20 次运行，发现输出层学习规则的进化结果中最佳者形式非常简单：

$$\Delta w_{ij}^{l} = \eta O_{jm}^{l} \varepsilon_{im}^{l} \tag{9.51}$$

将新发现的学习规则称为新学习规则(New Learning Rule, NLR)，这条规则实际上与著名的 δ 学习规则形式是相同的，δ 学习规则只能用来训练无隐层的神经网络。这里将隐层用 SBP 学习规则，输出层用新学习规则进行 XOR 神经网络的模拟实算，并将实算情况与只用 SBP 规则的情况进行比较，前者收敛速度大大提高，而且结果可靠性也有相应的提高。如图 9.40 所示，NLR 远比 SBP 要好，SBP + Rprop 收敛的学习次数为 110，而 NLR + Rprop 只有 40 次。在对其他测试问题的实算中，从类似的结果中也得到了验证。

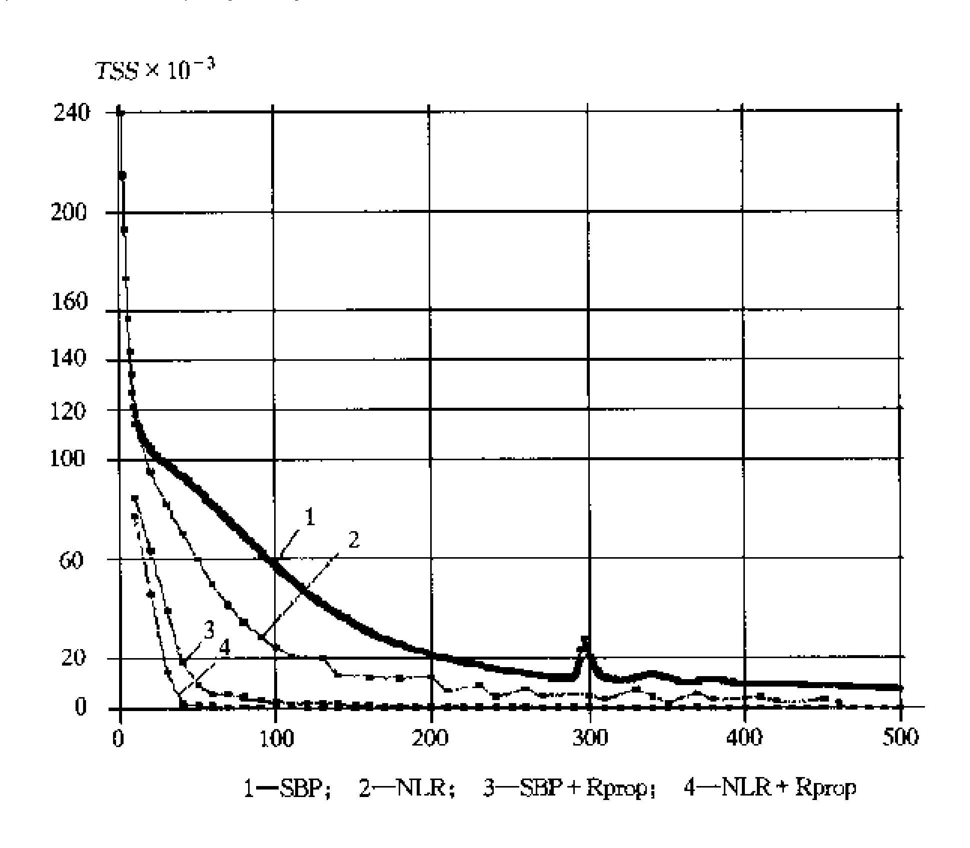


图 9.40 XOR 神经网络采用不同学习规则时 TSS 误差曲线

9.6 混合软计算技术

软计算(Soft Computing, SC)的概念是 L. A. Zadeh 在 1994 年提出的、并大力提倡的一种新技术，用来包容处理复杂的工业和商业系统诊断、评价和控制问题中有关非精确和非确定信

息、不完备领域知识的一整套计算技术。现有的软计算技术主要的包括模糊逻辑(Fuzzy Logic, FL)、神经计算(Neural Computing, NC)、进化计算(Evolutionary Computing, EC)和基于概率推理的计算(Probabilistic Computing, PC),研究中强调这些技术以各种形式相互融合、取长补短。其积极意义在于促进基于计算和基于符号相结合的各种智能理论、模型、方法的综合集成,以利于发展思想更先进、功能更强大、能解决更复杂系统的智能行为。通常,我们将这种互补的综合集成称之为混合软计算(hybrid soft computing),基于混合软计算技术的系统称为混合系统(hybrid systems)。混合系统中的技术集成,不论是属于一种松耦合还是属于一种紧耦合,均可以使原有方法系统的大部分结构得以保留,并且在系统的自适应性、最优性、健壮性以及计算和设计效率等方面得到提高。

混合软计算技术的构成如图9.41所示。图中将软计算技术分为两大类,一类是以概率计算和模糊计算为代表的近似推理技术,用于处理系统中的非精确和不确定信息以及知识的表达问题;另一类是以神经计算和进化计算为主的搜索和优化技术,其中神经计算同时属于一种非线性近似和隐含知识表达的技术。由于这两类问题都涉及到信息和知识表达及处理的合理粒度问题,所以有的学者从这个角度定义了粒度计算(granular computing)的概念。目前,这两类技术混合方式和技术途径非常之多,其中以模糊-神经混合方式的研究较早,研究成果上也比较成熟。近年来,遗传-模糊和遗传-神经混合方式的研究和应用非常活跃,在本章前面两节中已作了比较详细的介绍。遗传算法在这类混合系统的设计中存在以下特点:

① 遗传算法作为一种全局搜索和优化技术融入系统中,它本身不能表达知识,但具有较强的学习能力和优化能力,基于遗传算法的混合系统设计必须考虑能力互补的问题;

② 遗传算法的编码方法、遗传操作算子需要进行适当地调整;

③ 尽可能将现有的近似推理技术和优化技术有机地集成到混合系统中。

进化算法与模糊逻辑的结合是将知识获取和知识表示有机地结合起来的一种途径,目前研究的重点主要是用遗传算法改进模糊逻辑控制器。此外,模糊遗传算法(fuzzy genetic algorithms)借助模糊逻辑中的集合操作和关系算子设计新的遗传操作,用模糊逻辑控制的方法控制遗传算法的参数等,可以认为是两者之间的一种反向结合。遗传算法用来解决模糊优化问题、模糊神经控制器设计问题、模糊专家系统中知识库自动设计以及知识过滤问题,模糊信息检索等等也是值得注意的结合途径。

进化算法与神经计算的结合,可以使神经网络扩大搜索空间、提高计算效率以及增强神经网络建模的自动化程度。从粒度计算的角度,进化算法属于粗粒度计算(尽管可以细粒度实现),神经计算属于细粒度计算;从处理计算的角度来看,进化算法属于一种全局搜索优化方法,神经计算属于一种局部搜索优化方法。两方面的结合途径主要依靠进化算法确定神经网络连接权值、拓扑结构、学习规则的发现等,进化算法主要起优化作用。目前,神经网络结构的优化还存在许多的困难,有些神经网络模型的计算时间与神经元的多少关系并不大,却与学习的样本有明显的依赖关系。因此,有关非线性动力系统、混沌神经网络以及神经网络的数理研究的进一步支持就显得非常重要,有必要将进化并行算法与神经网络计算复杂性结合起来。此外,进化算法和神经计算的超高速实现也是重要的研究领域。

值得注意的是,进化计算、神经计算都属于生物计算的范畴,目前这一领域研究的个性化色彩越来越浓,人们会从越来越多的生物学发现的启发中研究新的计算方法和混合系统。例如1994年L. Adleman首次用实验显示了DNA用于计算的可行性,随后引起了许多学者尤其

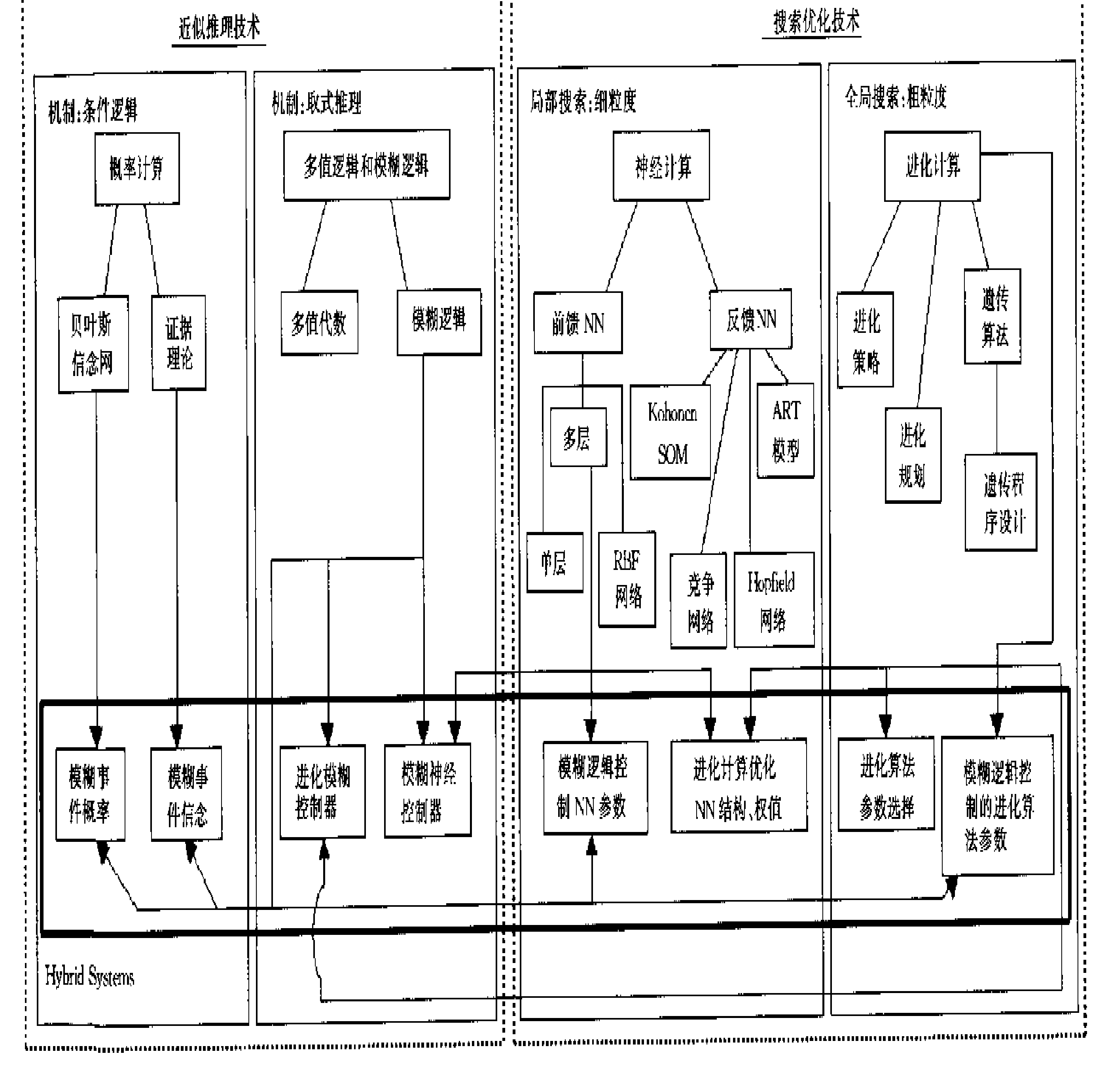


图 9.41 混合软计算的构成

是计算机科学家的兴趣，DNA计算的新颖性不在于其算法和速度，而在于可以采用迄今为止还没有作为计算机硬件的生物技术来实现，并且开发了这种媒体潜在的并行性。

DNA计算开始用于求解有向哈密顿路径问题(一个完全的NP问题)，后来用于美国国家安全署发明的数据加密标准系统(DES)的攻击实验，因它有约10^{17}种密钥，用于硅计算机已足够，但用DNA计算，不超过1 000步即可攻破此保密系统。有关将DNA计算方法与软计算方法相结合方面的研究已逐渐开展起来，例如，将常规遗传算法的二进制编码改进成DNA的ATCG编码，遗传操作中增加反转技术，模拟DNA修补系统功能，开发基于DNA机理的递阶遗传算法，同时研究基于DNA机理的学习方法，并用于模糊系统和神经网络的生成和优化设计。因此，有的学者称之为"DNA生物软计算"。这些发展趋势预示了软计算技术中随着生物技术的新发展而出现富有挑战性的成分将越来越多，其发展前景也越来越明晰化。

参考文献

[1] Zimmerman H J. Fuzzy Sets and Its Applications. Kluwer－Nijhoff Publishing, 1986

[2] Kosko B. Neural Networks and Fuzzy Systems. Prentice Hall, 1992

[3] Karr C L. Genetic Algorithms for Fuzzy Controllers. AI Expert, February 1991, 6(2): 26～33.

[4] Karr C L, Gentry E J. Fuzzy Control of pH Using Genetic Algorithms. IEEE Transactions on Fuzzy Systems, 1993 1(1): 46～53

[5] Karr C L. Designing Precise Fuzzy Systems with Genetic Algorithms. In: Herrera F and Verdegay J(ed)Genetic Algorithms and Soft Computing, Physica Verlag, 1996, 331～348

[6] Hashiyama T. A Study Finding Fuzzy Rules for Semi－Active Suspension Controllers with Genetic Algorithms: http://www.bioele.nuee.nagoya－u.ac.jp/member/tom/papers/icec/index.html

[7] Ng K C, Yun L. Reduced Rule－Based and Direct Implementation of Fuzzy Logic Control Systems. Research Report, Department of Electronics and Electrical Engineering, University of Glasgow, 1994

[8] Shimojima K. Unsupervised/Supervised Learning for RBF－Fuzzy System－Adaptive Rules, Membership Functions and Hierarchical Structure by Genetic Algorithm. In: Advances in Fuzzy Logic, Neural Networks and Genetic Algorithms, IEEE/Nagoya－University World Wisepersons Workshop, Nagoya, Japan, 1994, 127～147

[9] Leirch D D. A New Genetic Algorithm for the Evolution of Fuzzy Systems. Doctoral Dissertation, Robtics Research Group Department of Engineering Science, University of Oxford, 1995

[10] Gruau F. Genetic Micro Programming of Neural Networks. In: Genetic Programming, Koza J T(ed), 1994, 495～517

[11] Karr C L, Yeager D. Genetic Algorithms for Tuning Empirical Computer Models Used in Knowledge－Based Systems. 1993

[12] Filipic B, Juricic D. A Genetic Algorithm to Support Learning Fuzzy Control Rules from Examples. In: Genetic Algorithms and Soft Computing, . Herrera F and Verdegay J(ed)Physica Verlag, 1996, 345～352

[13] Herrera F, Lozano M, Verdegay J L. Tuning Fuzzy Logic Controllers by Genetic Algorithms. International Journal of Approximate Reasoning, 1995, 12(3): 115～123

[14] Furuhashi T, Nakaoka K, Uchikawa Y. A Study on Fuzzy Classifier Systems for Finding Control Knowledge of Multi－Input Systems. In: Genetic Algorithms and Soft Computing, Herrera F and Verdegay J (ed), Physica Verlag, 1996

[15] Geyer-Schulz A. Fuzzy Classifier Systems. In: Fuzzy Logic: State of the Art, Lowen R and Roubens M (ed), Kluwer Academic Publishers, Dordretch. 1993, 345~354

[16] Li Y, Ng K C. Uniform Approach to Model-based Fuzzy Control System Design and Structural Optimization. In: Genetic Algorithms and Soft Computing, Herrera F and Verdegay J(ed), Physica Verlag, 1996, 129~151

[17] Cord'on O, Herrera F. A Hybrid Genetic Algorithm Evolution Strategy Process for Learning Fuzzy Logic Controller Knowledge Bases. In: Genetic Algorithms and Soft Computing, Herrera F and Verdegay J(ed), Physica Verlag, 1996, 251~278

[18] Cord'on O, Herrera F. Identification of Linguistic Fuzzy Models by Means of Genetic Algorithms. In: Fuzzy Identification: A User's Handbook. Driankov D, Hellendoorn H, and Palm R(ed), Springer-Verlag, Berlin. 1997

[19] Cord'on O, Herrera F. A Three-Stage Evolutionary Process for Learning Descriptive and Approximative Fuzzy Logic Controller Knowledge Bases from Examples. International Journal of Approximate Reasoning, 1994(11): 1~15

[20] Fukuda T, Hasegawa Y, Shimojima K. Structure Organization of Hierarchical Fuzzy Model using Genetic Algorithm. Japanese Journal of Fuzzy Theory and Systems 1995, 7(5): 341~348

[21] Heider H, Tryba V, Muhlenfeld E. Automatic Design of Fuzzy Systems by Genetic Algorithms. In: Fuzzy Logic and Soft Computing, Bouchon-Meunier B, Yager R R, Zadeh L A(ed), World Scientific. 1995

[22] Paredis J. Coevolutionary Computation. Artificial Life 2, MIT Press, 1995, 355~375

[23] Paredis J. Coevolutionary Process Control. In: Proceedings of ICAVVGA'97, 1997

[24] Garzon M H. Biomolecular Computing and Programming. IEEE Transactions on Evolutionary Computation, 1999, 3(3): 236~248

[25] Chellapilla K, Fogel D B. Gaining Insight into Evolutionary Programming Through Landscape Visualization: An Investigation into IIR Filtering, Adaptive Distr. Parallel Computing Symposium, Dayton OH, 1996, 252~258

[26] Yao X. Evolving Artificial Neural Networks. In: Proceedings of the IEEE. 1999, 87(9): 1423~1439

[27] Chellapilla K. Evolution, Neural Networks, Games, and Intelligence. In: Proceedings of the IEEE, 1999, 87(9): 1471~1495

[28] Radi A, Poli R. Genetic Programming Can Discover Fast and General Learning Rules for Neural Networks. Technical Report CSRP-98-3, School of Computer Science, the University of Birmingham, UK, 1998

[29] Esparcia-Alcazar A I, Sharman, K C. Genetic Programming Techniques that Evolve Recurrent Neural Network Architectures for Signal Processing. In: Proceedings of the 1996 IEEE Signal Processing Society Workshop, Sixth in a Series of Workshops Organized by the IEEE Signal Processing Society Neural Networks Technical Committee, 1996, 130~139

[30] Lee J H, Hyung L K. Fuzzy Identification of Unknown Systems Based on GA. In: Simulated Evolution and Learning, First Asia-Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 216~223

[31] Hwang, M W, Choi J Y. Evolutionary for Projection Neural Networks. In: Simulated Evolution and Learning, First Asia-Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 146~156

[32] Cho S B, Lee S I. Hybrid Evolutionary Learning of Fuzzy Logic and Genetic Algorithms. In: Simulated Evolution and Learning, First Asia-Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 206~215

[33] Zhao Q. A Study on Co-evolutionary Learning of Neural Networks. In: Simulated Evolution and Learning, First Asia-Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 116~125

[34] Iba H, Kurita T, de Garis H, Sato T. System Identification Using Structured Genetic Algorithms in Genetic Algorithms. In: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgam Kaufmann Publishers, 1993, 279～286

[35] Ivakhnenko A G. The Group Method of Data Handling: A Rival of Stochastic Approximation. Soviet Automatic Control, 1968, 13(3): 43～55

[36] Kargupta H, Smith R E. System Identification with Evolving Polynomial Networks. In: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1991, 370～376

[37] Kristinsson K, Dumont G A. System Identification and Control Using Genetic Algorithms. IEEE Trans. SMC, 1992, 22(5): 1033～1046

[38] Moody J E, Darken C J. Fast Learning in Networks of Locally - Tuned Processing Units. Neural Computation, 1989(1): 281～294

[39] Parodi A, Bonelli P. A New Approach to Fuzzy Classifier Systems. In: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1993, 223～230

[40] Valenzuela - Rendon M. The Fuzzy Classifier System: A Classifier System for Continuously Varying Variables. In: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1991, 346～353

[41] Cordon O, Herrera F. A General Study on Genetic Fuzzy Systems. In: Genetic Algorithms in Engineering and Computer Science, . Winter G(eds), Wiley, 1995, 33～56

[42] Velasco J R, Magdalena L. Genetic Algorithms in Fuzzy Control Systems. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed), Wiley, 1995, 141～165

[43] Whitley D. Genetic Algorithms and Neural Networks. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed), Wiley, 1995, 203～214

[44] Karr C L, Gentry E J. Control of A Chaotic System Using Fuzzy Logic. In: Fuzzy Control Systems, Kandel A(ed), CRC Press, 1993, 476～496

[45] Bonissone P, Goebel K. Hybrid Soft Computing System: Industrial and Commercial Applications. In: Proceedings of the IEEE, 1999, 87(9): 1 641～1 665

[46] Paredis J. Coevolutionary Computation, Artificial Life 2, MIT Press, 1995, 355～375

[47] 山本透. 遺伝的アルゴリズムを用いたPID制御器の—設計. 計測自動制御学会論文集, 1999, 35(4) 420～429

[48] 平沢宏太郎. 遺伝的共生アルゴリズム, 計測自動制御学会論文集, 1999, 35(9): 1 198～1 205

[49] 長尾智晴. 遺伝的的手法による神経回路網の構造進化. 信学論, 1992, J75－D－II: 1 534～1 537

[50] 長尾智晴. 遺伝的的手法による任意結合型神経回路網の構造進化. 信学技報, 1992, NC92－20

[51] 特集. 遺伝的アルゴリズム. 計測と制御, 1993, 23(1): 1～81

[52] 田中一男. インテリジエント制御システム. 共立出版株式会社, 1996

[53] 田中一男. アドバンストフアジイ制御. 共立出版株式会社, 1994

[54] 孙增圻等. 智能控制理论与技术. 清华大学出版社, 广西科学技术出版社, 1998

[55] 阎平凡, 张长水. 人工神经网络与模拟进化计算. 北京: 清华大学出版社, 2000

[56] 张智星, 孙春在, 水谷英二. 神经－模糊和软计算. 张平安, 高春华 等译. 西安: 西安交通大学出版社, 2000

[57] 易继锴, 侯媛彬. 智能控制技术. 北京: 北京工业大学出版社, 1999

[58] 盛万兴, 戴汝为. 关于智能控制. 电子学报, 1999, 27(5): 86～89

[59] 程启明, 万德钧, 陈雪丽. 遗传学习算法的神经网络自适应船舶操纵控制系统研究. 模式识别与人工智能, 1998, 11(3): 305～309

[60] 金耀初，蒋静坪．基于遗传算法的模糊控制器分析．模式识别与人工智能，1997，10(1)：75～80
[61] 曹先彬，庄镇泉．一种基于遗传算法的模糊规则生成方法．模式识别与人工智能，1997，10(2)：171～175
[62] 郭了华，粟海华．基于神经网络和遗传算法的地图自动输入新算法．模式识别与人工智能，1998，11(2)：190～197
[63] 董聪，郭晓华．计算智能中的热点问题．计算机科学，1999，26(4)：5～9
[64] 胡炜，沈理．遗传优化模糊逻辑控制器．计算机科学，1997，24(6)：10～15
[65] 杨晓红，刘乐善．用遗传优化神经网络结构．计算机科学，1997，24(2)：59～65
[66] 孟祥武，程虎．进化神经网络编码表示机制研究．计算机科学，1998，25(1)：55～57
[67] 张晓缋，戴冠中，徐乃平．遗传算法在抽取和过滤模糊控制规则中的应用研究．控制理论与应用，1998，15(3)：379～384
[68] 雷佳，蒋静坪．基于遗传算法的模型参考自适应控制．控制理论与应用，1998，15(3)：466～468
[69] 张其光，王执铨．基于遗传算法的模糊神经网络控制器设计与稳定性分析．控制理论与应用，1999，16(5)：767～769
[70] 王晶，李玉兰，蔡自兴．基于遗传算法的模糊系统优化设计方法．控制理论与应用，1999，16(5)：699～704
[71] 廖俊，朱世强，林建亚．遗传算法在T－S模糊模型辨识中的应用．信息与控制，1997，26(2)：140～145
[72] 孟祥泽，刘新勇，车海平．基于遗传算法的模糊神经网络股市建模与预测．信息与控制，1997，26(5)：388～392
[73] 刘宝坤，石红瑞，王慧．基于遗传算法的神经网络自适应控制器的研究．信息与控制，1997，26(4)：311～314
[74] 刘永红．神经网络理论的发展与前沿问题．信息与控制，1999，28(1)：31～46
[75] 刘向杰．模糊控制研究的现状与新发展．信息与控制，1999，28(4)：283～291
[76] 朱幼莲，孟志浩，何世春．基于进化规划的自适应IIR滤波．数据采集与处理，1997，12(3)：167～170
[77] 仲春辉，王学军，陈亮．一种复杂模糊系统生成方法．数据采集与处理，1997，12(4)：251～255
[78] 杨兆升，姜桂艳．遗传思想在最优控制动态交通分配中的应用．系统工程与电子技术，1998(4)：117～121
[79] 张军英，许进，保铮．稳健二进前向网络的遗传训练方法．系统工程与电子技术，1999，21(2)：59～63
[80] 程启明，陈熙源，万德钧．基于遗传算法的模糊神经网络智能控制器及其应用．系统工程与电子技术，1999，21(8)：41～44
[81] 徐娟，汪懋华．并行遗传算法与神经网络、模糊系统的结合．小型微型计算机系统，1997，18(7)：1～7
[82] 倪远平，邹金慧，陈艾．改进型GA－BP训练策略与模式分类．小型微型计算机系统，1998，19(7)：40～44
[83] 赵明旺．非线性回归模型辨识的混合计算智能算法．系统工程理论与实践，1997(10)：99～103
[84] 范宏，王直杰，汤兵勇．基于遗传算法的神经网络销售量预测模型．系统工程理论方法应用，1998，7(3)：17～21
[85] 王以直，张志强，李敏强．杂合遗传算法与计算智能．系统工程理论与实践，1999(3)
[86] 孟祥武，程虎．基于遗忘进化规划的Hopfield网学习算法．软件学报，1998，9(2)：151～154
[87] 钟晓敏，方建安．基于遗传算法的混沌系统模糊控制．计算技术与自动化，1998，17(1)：1～3

[88] 周翔，蔡自兴．基于多分辨率遗传算法的多层感知神经网络及其在短期电力负荷预估中的应用．计算机工程与应用，1997(10)：10～12
[89] 唐淑琴、武颖琪．基于遗传算法的非线性参数估计器．仿真学报，1995，7(4)：43～46
[90] 赵凯，王珏．问题的基因－蛋白质表示模型．计算机学报，1998，21(9)：98～104
[91] 胡炜，沈理．一种进化模糊逻辑控制器的新方法．计算机学报，1999，22(6)：662～667
[92] 孟祥武．图文遗传算法．计算机工程与科学，1998，20(4)：14～16
[93] 刘尝，刘三阳，马建峰．基于遗传算法的多传感器数据融合．微电子学与计算机，1999(2)：22～27
[94] 徐洪泽，徐漫涛，张恩福．一种改进的遗传算法用于二自由度 PID 调节器设计．系统仿真学报，1998(2)：59～64
[95] 屈文忠，邱阳．基于遗传算法的多变量模糊控制器的设计方法．西安交通大学学报，1999，33(6)：101～103
[96] 盛兴华，赵俊慧．软计算及其新一代模糊控制器．北方交通大学学报，1999，23(2)：15～18
[97] 顾峻，沈炯，陈来九．遗传算法对模糊控制的优化及其应用．东南大学学报，1998，28(9)：109～118
[98] 黄景平，蔡骏，冯珊．基于 agent 的神经网络训练模式．华中理工大学学报，1999，27(3)：78～79
[99] 王宏伟，尤传富，张彤．基于浮点数编码 GA 的 T－S 模型的模糊辨识方法．吉林工学院学报，1998，19(3)：59～63
[100] 黄建军，谢维信．自适应模糊进化算法．电子学报，1997，25(10)：116～118

第 10 章　遗传算法与人工生命

生物的多种特性与功能已经给科学研究者很多启示，人工神经网络可以说是受到人的神经元的启发，而遗传算法、进化计算更是根据生物的遗传、进化机制演变而成的。近年来，一些学者对人工生成具有生命特征的产物进行了多方面的研究。在早期，冯·诺依曼根据许多小单元均只与其附近小单元相互作用的原则构造了元胞自动机，这种自动机可以“自繁殖”。他又指出计算机程序可以在内存中进行自复制。现在出现的各种计算机病毒，它们在一定条件下可以自繁殖，甚至“进化”。各种电子宠物如“电子鸡”、“电子鱼”等都具有类似生物的特性，例如需要按时进食，对环境可以趋利避害，有发育、生长、会生病、死亡等等生物特征。这些“人工生命”可以看成是一种具有生命特性的信息系统，相当一部分的人工生命是在计算机上实现的。可以认为自然界产生的生物是以核酸分子作为信息载体的，而生命的特征是由生物体中的信息系统决定的，如果把表征生命特性的信息系统在其他媒体上实现，同样也会产生相应的特征。这就是人工生命的内涵。

本章将首先介绍人工生命研究的形成与发展，以及一些基本观点、理论和方法。然后分别介绍人工生命模型两种不同的设计和实现方法，并分析进化中的突现行为现象；此外，将一般介绍与遗传算法相类似的免疫系统模型，以及进化硬件问题、进化对策论的研究和发展。

10.1　人工生命概述

10.1.1　人工生命的形成与发展

人工生命(artificial life，简称 ALife)的研究，可以追溯到 40 年代维纳的《控制论》以及薛定谔的《生命是什么》，50 年代冯·诺依曼的《自增殖元胞自动机理论》和 60 年代后期 A. Lindenmaker 提出的 L－系统(即植物形态学理论)以及 J. H. Holland 的遗传算法、J. Conwey 提出的“生命游戏”。80 年代开展的人工神经网络和混沌分形研究，均促使“人工生命”研究的确立。人工生命的概念则是美国国立洛斯·阿拉莫斯实验室 C. G. Langton 博士于 1987 年正式提出来的。1978 年他在学生时代发表了一篇题为“信仰的进化”的论文，其基本观点是：生物和文化的进化是同一现象的两个不同方面，文化的“基因”是信仰，信仰反过来又被记录在文化的基本“DNA”——语言上。后来在他对冯·诺依曼的未竟工作——自增殖元胞自动机(Cellular Automata，CA)研究发生了浓厚兴趣，元胞自动机假设每个元胞都是一个有限自动机，每个元胞有 29 种状态，并且状态随时间自动更新，状态转换规则依赖于当前状态和相邻元胞的状态，不同的规则会导致不同的行为。由于要求大量的元胞格，而且相邻元胞根据简单的规则进行相互作用有时会产生复杂的全局模式，它不能简单地从规则中预知。整个系统非常复杂，现有任何计算机难以胜任，但其研究的诱惑力很大，既然生命最基本的特征是自我增殖，机器能做到吗？ C. G. Langton 实现了具有八种状态(对应 $10^{30\,000}$个基因型)自我复制的元胞自动机，

他的方法只是嵌入了进化机制。后来又进一步发现了用CA模型能复制出动态演化过程中的吸引子、自组织和混沌现象。

人工生命研究主要集中在美国圣菲研究所(Santa Fe Institute, SFI),经过近10余年的发展,已经成为生物信息智能处理研究的一个重要方向,吸引了系统科学、计算机科学、人工智能、控制科学、生物科学、机器人学、经济学、哲学以及人类学等众多领域的专家学者投入研究。如今,在美国、欧洲以及日本,这方面的研究最为活跃。目前有三个专门的系列学术讨论会,它们分别是 Artificial Life,迄今已举办六届;欧洲的 European Conference on Artificial Life(E-CAL),迄今已举办四届,还有日本的 Artificial Life and Robotics(AROB)也已举办了四次。除此之外 IEEE 也曾召开一些内容基本相同的国际会议。

一般认为,人工生命试图在计算机等人工媒体上仿真、合成和生物有机体相关联的一些基本现象,如自我复制、寄生、免疫、竞争、进化、协作等,从"生命之如其所能"(life as it could be)的生物世界走向"生命之如吾所识"(life as we know it)的疆域(可以是非生物世界),研究生物智能的行为突现性和基于这些复杂行为的人工生命系统。

目前的人工生命系统有新生件(wet ware)、软件(software)和硬件(hardware)三类,其中新生件采用人工制造或控制组织分子产生的人工系统如克隆动物和器官;软件则依靠计算机模拟的生物系统,如人工虫、鸟群飞过障碍物的群体操作、鸟声模拟比赛等,以及模拟生物形态变化过程的工具有元胞自动机、L－系统等。如 Demetri Terzopoulos 等的人工鱼演示系统较好地在一个仿真的物理世界中演示了自律运动、感知、行为和学习,ALIFE 是一个比较完整的人工生命系统,其重点在于生物行为的动态仿真。硬件多见于人工整体控制的机器人,如 Steels 等的进化机器人原型系统。构造人工生命的方法通常分为弱方法(weak approach)和强方法(strong approach)。前者通过模拟已知的生命现象或生物组织功能建立人工生命模型,以加深理解实际的生命现象所蕴涵的复杂行为理论,如形态方面的新陈代谢、多细胞人工生命的进化、自组织自适应建模、细胞分裂、人工生物链等;后者试图制造人工生命,并探索生命的本质,它不局限于计算机程序,可以是任意的生命形式如机器人和人工大分子合成物。

10.1.2　人工生命的研究内容

从近年来国际上关于人工生命的研究内容分析,大致可以归纳为以下几个方面:

1. 数字生命的研究

所谓数字生命专指以计算机为工具和媒体、计算机程序为生命个体的人工生命研究,这方面以 T. Ray 的数字生命世界 Tierra 为代表。Thomas S. Ray 是一位生物学家、进化论学者,他把生物学上有机体进化的概念引入计算机领域,用数字计算机所提供的资源为他的数字生命提供一个生存环境。他设计的数字生命以数字为载体,探索进化过程中所出现的各种现象、规律以及复杂系统的突现行为。数字生命利用 CPU 时间来组织其在存储单元中的行为。数字生命以一定的计算机程序形式存在于 RAM 环境中,它为占据 CPU 运行时间、存储空间而通过响应的竞争策略相互竞争。一个"生命"必须被设计为适合在这样的环境中生存的某种数字代码程序。这个程序能够自我复制,并且直接被 CPU 运行,这些机器代码能够直接触发 CPU 的指令系统以及操作系统的服务程序,通过对资源的占有来体现它在进化过程中的优势地位。

在 Tierra 的运行中,随着世代的推移,生命体呈现出复杂的现象,种类日益增多,同时"单细胞"向多细胞进化,形成自己的生态环境。在生命的进化过程中,曾经出现过物种大爆炸的情况。如今,Tierra 运行于全球150个网络环境之中,其复杂程度还在不断增大。

数字生命的研究中一个重要的模型就是元胞自动机，元胞自动机被认为具有突现计算(emergent computation)功能。由于人工生命研究的重要内容是进化现象，遗传算法是研究进化现象的重要方法之一。

2. 数字社会的研究

Joshua M. Epstein 和 Robert Axtell 在计算机上创立了一个数字社会 Sugarscape。这个人工社会用来研究文化和经济的进化过程。他们认为一个人工社会是这样的计算机模型，它包含：一群具有自治能力的行为者；一个独立的环境；管理行为者之间、行为者与环境之间以及环境各个不同要素之间相互作用的规则。人工社会的行为者是一个能够随着时间发生变化或者具有适应性的数据结构。每个行为者具有遗传特性、文化特性，以及管理它与环境和其他行为者之间的规则。其中行为者的遗传特性在其生命周期内是固定的。在 Sugarscape 中行为者的性别、新陈代谢以及视野是其遗传特性；而文化特性是由父母传给子女，并通过与其他行为者的联系而横向地发生改变。其环境里含有可更新的能源——糖，行为者依赖糖组织自己的新陈代谢。人工社会是由各个行为者自我组织形成的，由各个行为者在简单规则的支配下，与人工环境交互作用突现形成的。

3. 虚拟生态环境

挪威的 Keith Downing 提出了名为 EUZONE 的一个进化的水中虚拟生态环境，目的是提供一个观察生态系统是如何从原始状态进化以及复杂生态系统突现行为的实验手段。它利用具体的物理和化学模型，结合进化规划建构以碳元素为基础的水中生态环境，可以观察到低等动物形体的进化及生存竞争。EUZONE 具有两个基本过程：环境模拟和生物的进化，前者尽量反映真实世界的物理、化学以及生物之间的相互作用。生物进化由遗传程序设计和遗传算法来实现。

4. 人工脑(Artificial Brain)

日本的 ATR 的进化系统部(evolutionary system department)致力于开发新的信息处理系统，这种系统具有自治能力和创造性，他们将这样的系统称为"人工脑"。人工脑不仅能自发地形成新的功能，而且能够自主地形成自身的结构。其研制者并不想单纯地再现生物大脑的功能和结构，而是要得到在某些方面优于生物大脑的信息处理系统。

人工脑采取两方面的实现方式：类似生命的模型(life - like modeling)和社会模型(social modeling)，包括传统的用于神经系统的学习模型(如人工神经网络)。在类似生命的模型中系统有一个类似于生命系统胚胎发育的功能，使得系统的结构和组成单元能够发生变化，形成复杂系统。在社会系统中系统被视为动态过程，在这个过程中，局部的、各个单元之间的连接使得整体的、全局的功能及次序、状态发生突现。反过来，各个单元也受到全局状态的影响。因此两个方向的相互连接，影响系统发生变化。为使系统具备自治和创造性，系统本身需要一些机制在功能和结构上的自发变化。研制者在系统设计中引人"进化和突现"的极值。

ATR 对于人工脑的建构正在从硬件和软件设计两方面来推进，这个项目在数字计算机上通过自然选择的简化来产生复杂和智能化的软件。进化的基本因素是带有可遗传变异的自我繁殖。为了在硬件上实现进化计算，需要一种特殊的硬件平台，可进化的硬件目前处于开发初期阶段。大规模的神经网络和极高速度要求，需要高容量的存储器以及高速的电子器件，CAM-Brain 项目运用"进化工程"(evolutionary engineering)技术来建构、发展、进化出以 RAM 和元胞自动机为基础的人工脑。

5. 进化机器人(evolutionary robotics)

生物系统给人们提供了分布式控制的思路,其脑神经系统、遗传系统、免疫系统的功能启发了人们把生物学上的一些现象工程化,并且运用于机器人的设计上。目前正在发展的第三代机器人要求具有人的简单智力和学习能力。

Rodney A. Brooks提出了基于行为的设计方法。此方法在20世纪80年代中期开始使用,设计出比传统设计方法行动更快和更灵活的机器人。进化机器人的操作方式是自律型的,其位置、移动等是突现形成的,其智能也是由各个并行执行的小过程自组织突现形成的,并且这样的小过程分散在整个系统中。进化机器人具有比传统机器人更快的速度和更好的灵活性、鲁棒性,进化计算可以比较容易地植入到这样的系统中,其硬件和软件的设计以及测试费用比以前要少。

6. 进化软件代理(evoluable multiagent)

人工智能的研究人员长期以来一直在研究一种复杂的建构软件代理的方法。例如,一个具有入工智能的电子邮件Agent可能知道有行政助理人员,知道某位用户有一名叫George的助理,知道助理必须掌握老板的会议日程,还知道"会议"这个词的信息可能含有日程信息。有了这些知识,这个Agent就可以推导出它应当转送此信息的复制件。

以知识为基础的软件代理要求包括所有常识信息的知识库,但一般软件工程师只能系统地整理比较狭窄领域的知识。采用人工生命的方法进化软件Agent可能是最有发展前途的方法。"人工进化"可以随着时间的推移整理出一个系统中的最有效的代理人(由其主人评定)的行为,并把这些行为结合起来以培养出适应能力更强的群体。可以设计这样的电子邮件Agent,它们能够连续观察一个人的行动,并把他们所发现的任何有规律的行为实现自动化。电子邮件代理观察到用户总是把含有"会议"的信息的复制件转交给行政助理,由此便可领悟到其规律,然后自动做这项工作。此外,Agent可以向执行同一任务的Agent学习,例如,一个电子邮件代理在遇到一份陌生的信件时可以询问它的同伴,从而得知人们通常是看了私人递交给他们的电子邮件后,再看按邮送名单递交的电子邮件。这类合作可以使一群代理以复杂的、明显智能的方式行动。

将人工生命Agent置于新一代计算机网络中,形成一个电子生态系统。对用户有用的或对其他Agent有用的Agent将运行比较频繁,从而得以生存下来并繁殖后代,那些用处不大的Agent将被清除掉。随着时间的推移,这些数字化生命形式将占据不同的生态环境。有的Agent可能进化成优秀的数据库编制者,其他的Agent则是使用它们的索引来找到某一用户感兴趣的文章。可能会出现寄生、共生、免疫以及生物世界中常见的其他现象在计算机网络中的实例。随着外部对信息的要求发生变化,这个软件生态系统将连续地更新自身。

IBM公司目前正在开发一种被称为电脑空间免疫系统的软件。正如脊椎动物的免疫系统在一种新病原侵入机体后几天之内就会产生出对付它的免疫细胞一样,计算机免疫系统可在几分钟内就能产生出识别并消除新遇到的计算机病毒的方法。

10.1.3 人工生命研究的特点

许多生物模型一般假定种群中的个体是相同的,或者用种群中个体的均值来描述群体现象。因此,生物模型中同一种群中个体的交互作用也是同一的。尽管如此,基于个体的生物模型不可避免地需要强调个体差异和局部交互作用。生物分布模型和形态模型是两类基本的个体生物模型,生物分布模型认为种以群分、物以类聚;形态模型按个体差别和局部交互对种群

中所有个体进行考察。个体形态模型可以看成是一种人工生命模型，人工生命的创始人 C. G. Langton 认为人工生命模型区别于个体模型有四个方面：

① 人工生命模型由多个描述个体行为的简单程序组成；

② 不存在一个程序起主导作用；

③ 每个程序描述个体在局部环境中的交互方式；

④ 不存在确定种群整体行为的规则，任何高于个体级的群体行为都是突现的。

Langton 认为人工生命采取自下而上的方法，从组成系统的自主个体开始，由个体的相互作用，逐渐进化、发展，从而实现突现的复杂行为。

种群的某种生命现象可以从个体行为中突现出来，如物种普遍存在的所谓"镶嵌进化"(mosaic evolution)，主要是物种某些器官形态或功能在强的选择压力作用下突现出来，呈现快速进化，使得器官形态或功能进化程度不一。突现(emergence)(有涌现、创发等几种说法)是人工生命中的重要特性，关于突现行为目前尚没有统一的定义。在系统复杂性研究中，突现是一种源于低级行为和规律的、未预期的高级现象的呈现，例如，在元胞自动机的研究中，相邻元胞根据简单的规则进行相互作用有时会产生复杂的全局模式，它不能简单地从规则中预知。Langton 认为突现是处于低级与高级行为之间的反馈，"在设定的边界条件下，一组个体局部地按动力学规律相互作用可以孕育出一种整体范围上的动态结构，并获得稳定。这种整体范围结构的构建既有局部的基础，又在进化中得到整体上的提炼"。

10.2 基于遗传算法的人工生命模型设计

遗传算法、遗传编程和进化计算等是人工生命系统开发的有效工具。一般而言，遗传操作过程和进化计算机制非常适合于描述人工生命系统。将环境、种群等因素包含在一个动态仿真框架中动态考察，基因型依据遗传规则完成其操作过程，在基因型到表现型的非线性映射下，表现型在时间坐标下的形态呈现动态变化，在群体与环境的交互作用的进行中，行为突现决定了进化的性状和方向。下面给出一个简单的人工生命模型设计原型。

10.2.1 人工生命的虚拟世界

假定虚拟世界为一二维矩形区域，只存在两种生物形式：生物 1 和生物 2。生物在虚拟世界中交配生成子孙，并假定两种生物均为雌雄同体，异种之间禁止交配。在虚拟世界中随机地生成植物食物，生物依靠植物食物和动物食物(生物的遗体)维持生命。每个生物个体都具有一定的能量，若获取食物自身能量增加，在一时间内，随着个体移动、攻击等行动的发生能量减少，各个体的移动模式、行动模式、寿命等作为遗传基因的表现型描述，若个体经过一定时间能量耗尽或寿命完结，该个体即自动死亡变成动物食物。此外植物食物与动物食物都具有有效时间的属性，超过有效时间便自动消失。如图 10.1 所示，分别用◇、□、△、 、▲ 表示生物 1、生物 2、植物食物、障碍物、动物食物。障碍物静止存在，生物、食物与障碍物不能占据同一位置。

在虚拟世界中，两种生物的个体为获取食物相互竞争，时而相互攻击，时而交配产生后代，种群随着时间演变，虚拟世界的环境如障碍物的设置、食物的分布等也在变化，通过世代的模拟，目的在于考察生物群的基因型，获得进化的性状和方向。

个体的遗传操作包括选择、交配和变异与常规的遗传算法稍有差异，必须选择在行动范围内的局部对象个体交配。个体的适应度不能直接由个体的表现型或基因型计算。世代交替时，某个体在生物群中是否具有生存权由它与其他个体的关系来决定，如果不对个体总数做限定处理，在严峻的生存条件下(即淘汰压力很强的情况下)存在全体生物群灭绝的可能。

下文中的虚拟环境及生物常数按自然特征相似性原则预先设定，因此不再作特别的解释。

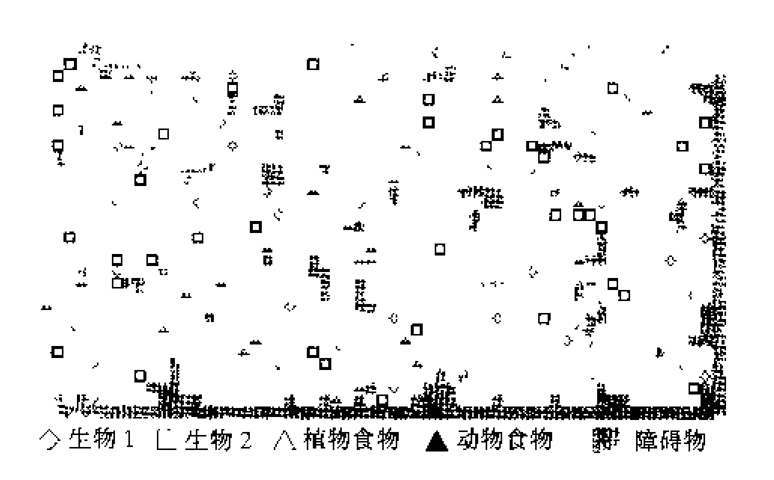


图10.1 人工生命的虚拟世界

10.2.2 个体的基因型

个体的基因型 G_i 可以用下式表示：

$$G_i = G_i(SP_i, SL_i, VF_i, TM_i, CM_i, LM_i, CA_i, CR_i, SA_i, DA_i, LA_i, EF_i) \quad (10.1)$$

这里，SP_i——生物种号，取0或1，位长1，$SP_i=0$ 属于生物1，$SP_i=1$ 属于生物2；

SL_i——寿命，取0～15，位长4，寿命 $= SL_MIN + SL_i$，SL_MIN 为最小寿命；

VF_i——视野，取0～3，位长2，视野 $=(2\times(VF_i+1)+1)^2$，最小 3×3，最大 9×9；

TM_i——移动模式，取0～3，位长2，

如图10.4所示，$TM_i=0,1$ 时取基本模式1，

$TM_i=2$ 时取基本模式2，

$TM_i=3$ 时取基本模式3；

CM_i——移动特性，位长3，第1位：向同种生物移动(0：负，1：正)，

第2位：向异种生物移动(0：负，1：正)，

第3位：向食物移动(0：负，1：正)；

LM_i——移动消耗，取值0或1，消耗能量 $= LM_i+1$；

CA_i——行动特性，取值0～7，位长3，$CA_i=0,1,2$：(1,2,3)，　$CA_i=3$：(1,3,2)，

$CA_i=4$：(2,1,3)，　$CA_i=5$：(3,1,2)，

$CA_i=6$：(2,3,1)，　$CA_i=7$：(3,2,1)；

CR_i——善变率，取值0～7，位长3，不遵从行动特性的概率 $= CR_i/7$；

SA_i——攻击速度，取值0～7，位长3；

DA_i——防御能力，取值0～7，位长3；

LA_i——攻击消耗，取值0～7，位长3；

EF_i——食物获取效率，取值0～15，位长4。

生物个体还具有能量和年龄的属性，个体的能量 $Engy_i$ 取值0～100；个体的 Age_i 随着世代增加而增长，一旦超越寿命即自然死亡。对于食物而言，给定相当于寿命的常数作为限制时

间，植物食物限制时间 $TL_1=20$，动物食物限制时间 $TL_2=5$，因此食物的属性为新鲜程度 $Frsh_t$，随着世代增加而增大，超过其限制时间即消除。

10.2.3 个体的移动与行动

（1）**移动** 每个个体都有一个方形视野，如图 10.2 所示，视野大小为 $(2\times(VF_t+1)+1)^2$，视野范围内个体、食物和障碍物的分布可以推知。各个个体根据视野内的分布及其移动特性来决定如何移动。移动特性表明了对个体和食物的偏好，用正负号表示。例如（+，－，+）表示该个体喜欢接近同种个体和食物，不喜欢接近异种个体。建立如图 10.2 所示的坐标系，以该个体位置作为坐标原点，若视野范围内同种生物 $S_i(sx_i, sy_i)$ 有 N_s 个，异种生物 $D_j(dx_j, dy_j)$ 有 N_d 个，食物 $F_k(fx_k, fy_k)$ 有 N_f 个，该个体的移动特性为 (sgn_1, sgn_2, sgn_3) $(sgn_m=-1$ 或 $1)$，下式可计算移动目的点的坐标 (x_t, y_t)。

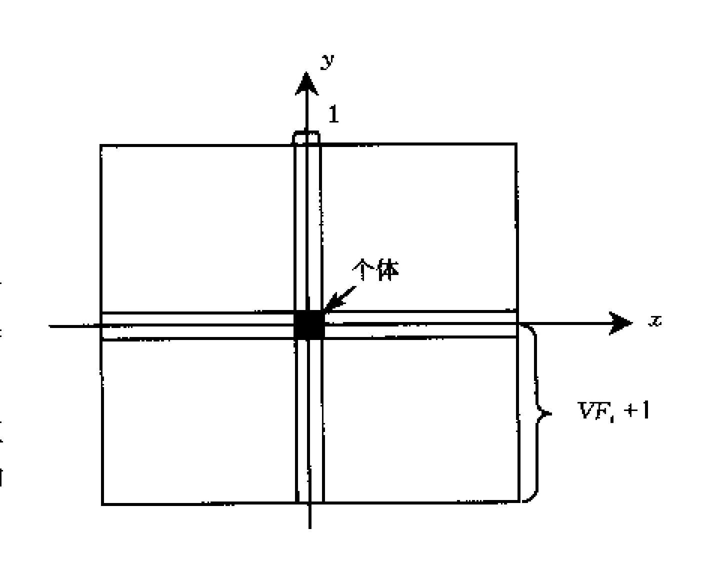


图 10.2 个体的视野

$$x_t=(sgn_1\sum_{i=1}^{N_s}sx_i+sgn_2\sum_{j=1}^{N_d}dx_j+sgn_3\sum_{k=1}^{N_f}fx_k)/(N_s+N_d+N_f) \tag{10.2}$$

$$y_t=(sgn_1\sum_{i=1}^{N_s}sy_i+sgn_2\sum_{j=1}^{N_d}dy_j+sgn_3\sum_{k=1}^{N_f}fy_k)/(N_s+N_d+N_f) \tag{10.3}$$

如图 10.3 所示从 8 个方向的移动向量中选择与 (x_t, y_t) 最近的向量，该向量的终点即成为该个体新的位置，个体的能量随之减少 LM_t+1。

当个体视野内无其他个体或食物时，遵照该个体的移动模式确定移动位置，如图 10.4 所示。$TM_t=0, 1$ 取基本模式 1，个体可以从相邻的 8 个位置中挑选任意一个位置移动；$TM_t=2$，取基本模式 2，个体可以从上下左右方向相邻的 4 个位置挑选任意一个位置移动；$TM_t=3$，取基本模式 3，个体可以从对角方向相邻的 4 个位置挑选任意一个位置移动。

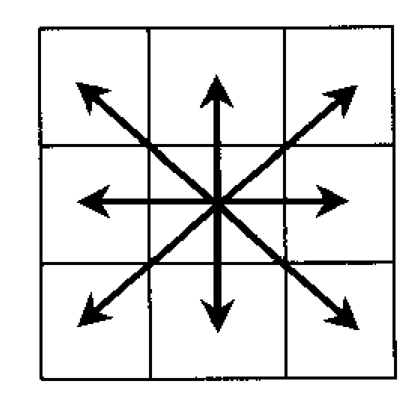


图 10.3 8 个方向的移动向量

（2）**行动** 在个体的行动影响范围内，存在其他个体或食物采取“行动”的条件如下：

对于攻击：范围内至少存在一个同种或异种生物个体；

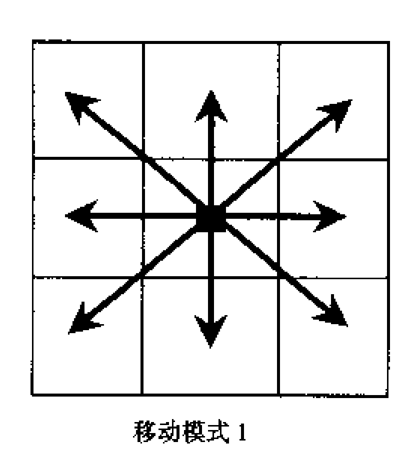
对于获取食物：范围内至少有一个食物存在；

对于交配：范围内至少有一个同种个体。

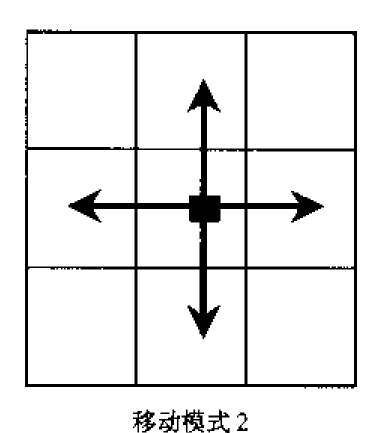
这些行动条件存在多数满足的情况下，按照个体的行动特性 CA_t 中的行动顺序进行选择，对于行动对象存在多数的场合，以概率 $(1-CR_t/7)$ 随机地选择其中一个即可。

伴随行动个体的属性将发生下列相应的变化：

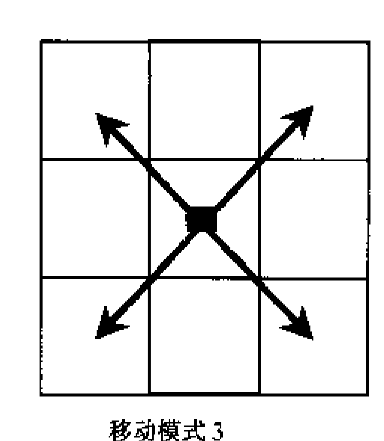
（1）**攻击** 在行动范围内，若个体 i 的攻击力大于个体 j 的攻击力，个体 i 获胜，个体 i 和



移动模式 1



移动模式 2



移动模式 3

图 10.4　三种移动模式

个体 j 的能量均减少，但减少量存在差异。

$$Engy_i(t+1) = Engy_i(t) - LA_i \times random \tag{10.4}$$

$$Engy_j(t+1) = Engy_j(t) - 40 - LA_j \times random \tag{10.5}$$

个体的攻击力按下式计算：

$$Attack_i = Engy_i(t) + SA_i \times 20/7 \times random + DA_i \times 20/7 \times random \tag{10.6}$$

（2）**获取食物**　个体 i 在行动范围内获取食物，能量增加为：

$$Engy_i(t+1) = Engy_i(t) + 40 \times (50 + EF_i \times 50/15)/100 \tag{10.7}$$

若 $Engy_i(t+1) > 100$，则令 $Engy_i(t+1) = 100$。

（3）**交配**　个体 i 在行动范围内与个体 j 发生交配，各自的能量减少 5 之后产生一个子个体，子个体的位置在两亲行动范围的逻辑并集内随机选择一点，子个体的基因型根据遗传操作实行均匀交叉生成，其能量属性为 100，此外，除了生物种号 SP_i 外其他基因型随机地以低概率产生变异。

10.2.4　世代演变的模拟过程

第 1 步　建立和设定虚拟世界，障碍物位置、植物食物随机地设定，每隔一定时间随机产生一定数量植物食物。

第 2 步　随机地产生初期种群个体的基因型，各个体的能量属性 $Engy_i$ 在 70～99 之间随机产生，年龄属性 Age_i 在 0～(SL_MIN-1)之间随机产生。

第 3 步　对于两种群的个体分别编号，个体的总数随世代演变而变化。

第 4 步　按个体编号顺序选择个体的状态变化。状态变化可以任意决定“移动”或“行动”：

移动：个体由当前位置改变到其他位置；

行动：包括攻击、获取食物和交配，可以任意选择其中之一。

在个体行动范围内，不存在其他个体或食物时，应该选择“移动”。

第 5 步　各个体的年龄属性 Age_i 加 1，将超越寿命$(=SL_MIN+SL_i)$的个体从生物群中消除，变为动物食物。

第 6 步　食物属性 $Frsh_i$ 加 1，消除超过限制时间的食物。

第7步　世代数加1，若超过指定演变代数，或者个体全部灭绝的情况下终止计算；否则，返回到第3步。

根据上述人工生命模型，我们实现了一个人工生命演示系统的实例，虚拟世界既可以文件设定，也可以完全随机产生；个体的遗传表现型包括生物种号、寿命、视野、移动模式、移动特性、移动消耗、行动特性、善变率、攻击速度、防御能力攻击消耗、食物获取效率，基因型长度32，指定演化代数120代。设定不同的初始条件和个体数以及初期个体的遗传基因型，考察虚拟环境中生物在进化过程中的演变规律，如可比较每代种群中最佳基因型，两种生物总数的世代变化，生物种群在虚拟环境中的分布变化等。图10.5所示为实现的人工生命演示系统在演变到第120代时的情况，包括虚拟世界、最佳基因型的个体及生物个体个数。此时生物1在虚拟环境中因难以维系已灭绝；生物2的个数由初始代的51增加到78，并且集中分布于虚拟环境之一隅。从生物2的最佳个体基因型分析，可知最佳个体的表现型具有以下的优越特征：

寿命 SL_i：长；视野 VF_i：大；移动模式 TM_i：0；移动特性 CM_i：(1，1，0)；移动消耗 LM_i：小；行动特性 CA_i：7；善变率 CR_i：小；攻击速度 SA_i：快；防御能力 DA_i：强；攻击消耗 LA_i：小；食物获取效率 EF_i：高。

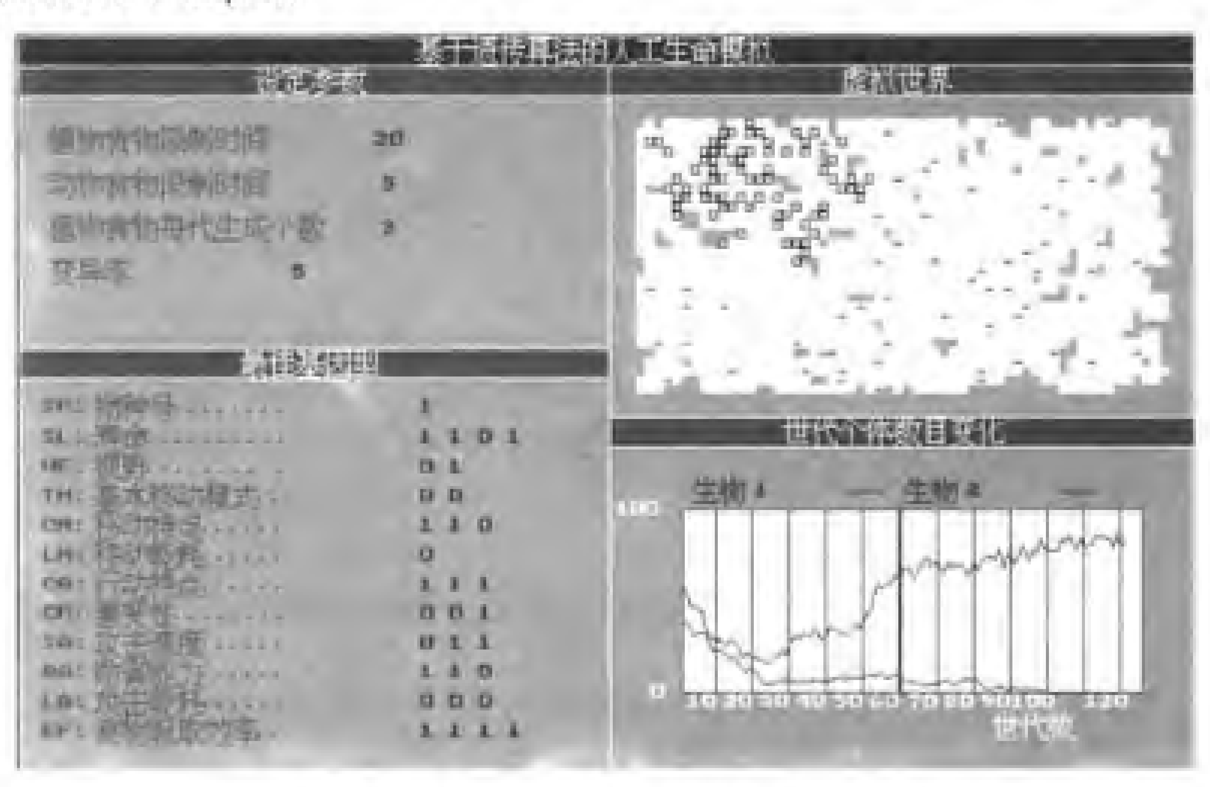


图10.5　人工生命演示系统　（第120代演化结果）

人工生命模型模拟的源程序参见附录II－4。

应该说，这里给出的人工生命模型是一个基本的研究实例，遗传基因的定义和解释都预先完全明确，与描述自然生命的多样性及其赖以生存的复杂环境和自然法则相比要简单得多。

10.3 蚁群协作觅食模拟模型

昆虫世界中,蚂蚁的组成是一种群居的世袭大家庭,我们称之为蚁群。之所以将蚁群比拟为一个家庭,是因为蚁群中除了亲缘上的互助关系外,成蚁划分为世袭制的蚁王(后)和工蚁两个等级,蚁群的大小从几个到几千万个不等,蚁群又具有高度组织的社会性,彼此沟通不仅可以借助触觉的、视觉的联系,在大规模的协调行动(集体行为)上可以借助外激素(pheromone)之类的生化信息介质。虽然单个蚂蚁的行为极为简单(大致有42种可分辨的行为),但由单个简单的个体组成的群体却表现出及其复杂神奇的行为,因此蚁学家们将整个蚁群视为一个功能强大的超级生物。大多数蚁类的生命周期是在蚁穴中完成的,而工蚁的觅食行为是最易观察的。工蚁捕食不是在发现食物后立即进食,而是将之搬回蚁穴与其家庭成员分享。每个工蚁具有如下的职能:平时在巢穴附近作无规则行走,一旦发现食物,独自能搬的就往回搬,否则就回巢搬兵;一路上它留下外激素嗅迹。其强度与食物的品质和数量成正比;若其他工蚁遇到嗅迹,就会循迹前进,但也会有一定的走失率,走失率与嗅迹的强度成反比。因此,蚁群的集体行为便表现出一种信息正反馈现象:某一路径上走过的蚂蚁越多,则后来者选择该路径的概率越大,蚂蚁个体之间就是通过这种信息的交流达到搜索食物的目的。人们在试验中发现,在离蚁穴等远的不同方位上各放一堆品质和数量大体相同的食物,则较小的蚁群搬运食物时两路兵力大体相同;而足够大的蚁群则会先集中兵力歼灭二者之一,只有少数兵力对付另一堆食物。这种做法在客观上是符合蚁群整体利益的。因为在同一地区可能还有其他蚁群与之争食。

工蚁的觅食过程可以分为搜索食物和搬运食物两个环节,觅食过程的成本主要发生在搜索环节,而觅食过程的收益主要发生在搬运环节,成功的觅食策略是最小化搜索食物的时间。Johnson, Hubbel 和 Feener(1987)构造了一个觅食过程的模型,研究食物堆的大小和空间分布对于工蚁的数量演变和觅食行为方式的影响。模型中假设在无增援下的食物搜索是一种在巢穴附近的随机行走行为。如果食物堆很小,工蚁的觅食是在无增援下独立进行的,因为增援的收益比较小。食物堆很大时,蚁群需要足够大的觅食力量,这主要地依赖于增援。增援的收益相当大,足以消除搜索的成本。增援的形式可以有多种,最简单的有长列队增援方式和成组增援方式以及大规模增援方式。大规模增援一般出现在有大堆食物被发现的场合。

在图10.6中,假设以下的模拟环境:

① 在每个网格上包含有蚂蚁数目以及是否存在巢穴、食物或者外激素的信息。外激素由蚂蚁移动时生成聚集,并随时间逐渐消失。

② 蚁群世界中可以分布数百个到数千个蚁群,每个蚁群包括十几个到数百个遗传上等价的个体,它们的觅食行为由神经网络来模拟,工蚁个体除了能够感知和搬运食物外,还能够感知和释放外激素,

③ 蚁王(后)的繁殖能力取决于工蚁搬运回巢穴的食物数量。蚁群进化模拟是以蚁群为进化对象,而不是单个的蚂蚁个体,因此每个蚁群都有一个基因编码与之对应,蚁群的适应度是工蚁搬运回巢穴的食物数量的函数,越有效的觅食行为意味着越大的再生能力。

④ 单个的蚂蚁个体虽然遗传上等效,但接受的信息不同,因此行为上各有差异,寻求集体的觅食行为策略来源于群体间生存的选择压力。

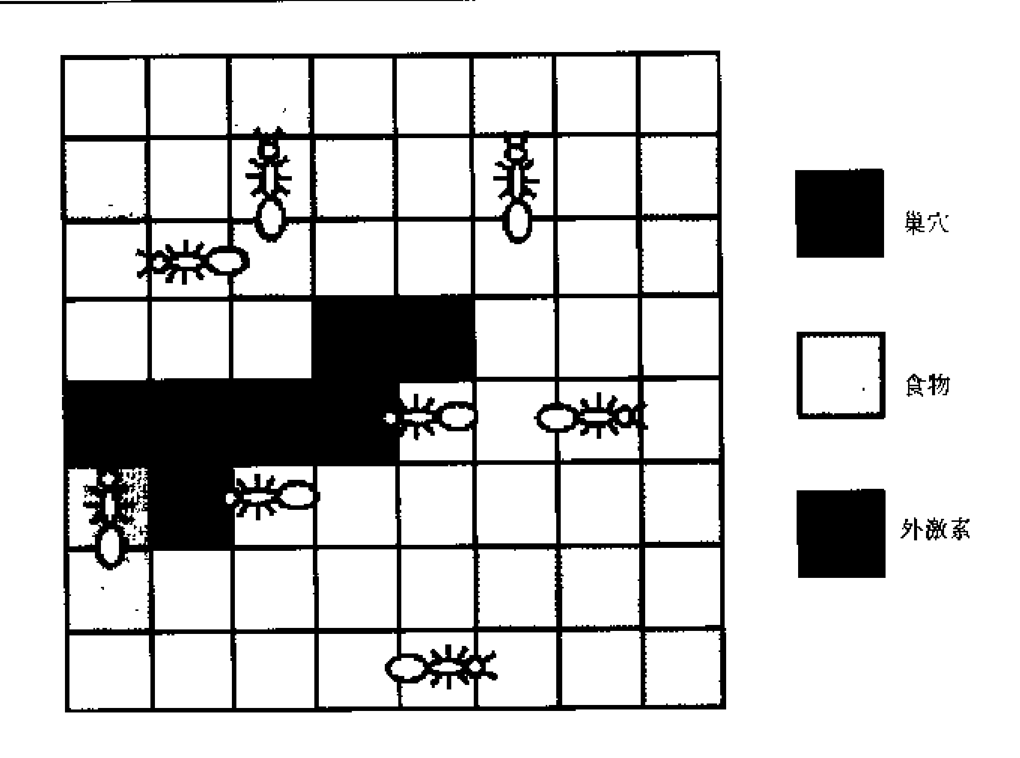


图 10.6 人工蚁群世界

⑤ 进化过程的每一代包括固定的时间，在每一时刻，每个蚂蚁个体必须感知其周围是否有巢穴、食物或者外激素的存在，接受这些信息来更新其状态，更新状态的行动包括移动、取食物、放置食物、释放外激素。蚁群范围内蚂蚁个体的状态一个接一个地改变，多个蚁群以并行的方式按照这种方式演化。

⑥ 模拟过程中蚂蚁个体之间不存在直接的交互作用，所有蚂蚁的个体交互依赖于环境中的变化。由于在一个共有的环境中，蚁群之间的交互实际上是对食物的竞争，所有的蚁群丢弃相同的外激素，因此蚁群之间交互更加直接。

在进化模拟的每一代，人工蚁群环境重新初始化，所有释放的外激素被取消，食物按一定概率重新分布，所有的蚂蚁个体返回巢穴进行新一轮的觅食行动，每个蚁群的适应度得分按照上一轮觅食成果减去该蚁群所有个体新陈代谢的能耗（包括移动、释放外激素等）计算。

将单个蚂蚁的觅食行为用反馈型神经网络来表示（如图 10.7 所示），采用合适的编码方式将神经网络信息映射为蚂蚁的遗传编码。蚁群内蚂蚁的遗传编码是一致的，但蚂蚁在某时刻接受环境激励不同，产生的行为各异。每一代计算各蚁群的适应度，完成蚁群的遗传操作包括选择、交叉和变异。经过若干代的进化，考察蚁群的觅食行为演变特征。

蚂蚁的觅食模式由感知输入映射为输出行为，采用神经学习的方式获得。感知的信号包括：食物、巢穴以及外激素数量的信息，此外搬运状态、搬运方向以及随机噪声也作为输入信号处理。输出行为包括：下一时刻移动的位置、是否搬运以及外激素释放数量。编码方式采用图 10.8 所示的按结点连接关系和连接权重进行组合描述，每一个权连接包括始结点神经元号和终结点神经元号及其连接权重三部分的二进制码，所有权连接编码组合起来成为神经网络的遗传编码。设定每个神经元活性变量用 0，1 表示，连接权用整数表示，隐层神经元的活性和输出层的输出按照线性求和的阈值函数计算，若加权和大于 0 取 1，否则取 0。表 10.1 给出了试验的神经网络设计方案，神经元数为 64 个，存在连接为 1 596 个。

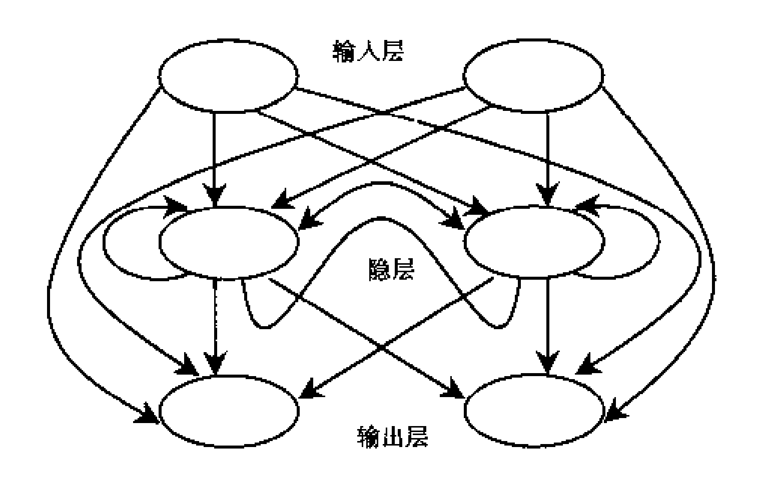


图 10.7　反馈型神经网络

基因编码

00110111110000110001010110001110000110001100110111010110000110000100100

结点 i	结点 j	连接权重
1(001)	5(101)	-2(1110)
0(000)	6(110)	2(0010)
5(101)	4(100)	7(0111)
0(000)	3(011)	1(0001)
4(100)	6(110)	-2(1110)
5(101)	4(100)	3(0011)
0(000)	2(010)	4(0100)

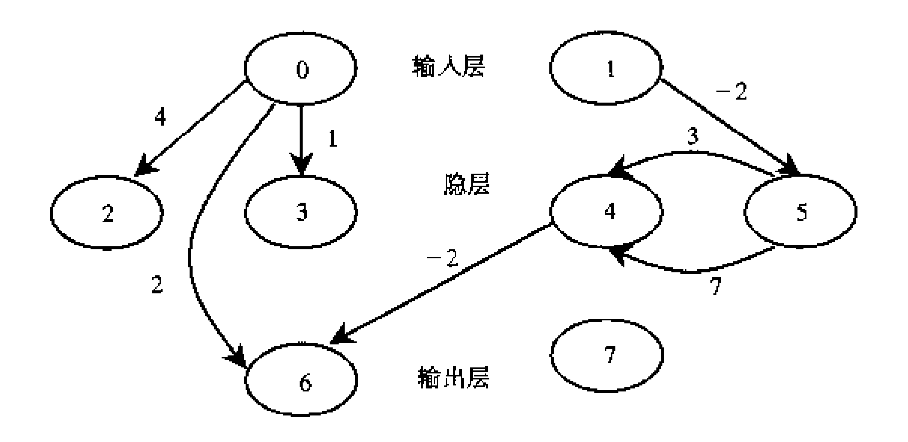


图 10.8　神经网络遗传编码

表 10.1 试验的神经网络设计方案

输入层	结点数
周围外激素的数量	9
食物位置	9
巢穴位置	9
巢穴的最近方向	4
随机噪声	4
搬运状态	1
隐层	21
输出层	
行进方向	4
是否搬运	1
外激素释放量	1
是否存放食物	1
神经元总数	64
权连接数	1 596
二进制染色体长度	(3+3+6)×1 596

结点号和连接权值分别用 3 位和 6 位二进制码表示，则该神经网络的染色体长度为(3+3+6)×1 596=19 152。

上述蚁群觅食行为的模拟模型可以称之为蚁元系统(Ant System)，蚁元系统实际上是群体智能(swarm intelligence)的一种模拟，单个蚂蚁缺乏智能，但蚁群则表现为一种有效的智能行为。"三个臭皮匠，顶个诸葛亮"是对个体智能与群体智能的最形象的说明，群体智能可以在适当的进化机制引导下通过个体交互以某种突现的形式发挥作用，这是个体以及可能的个体智能难以做到的。

10.4 免疫系统模型

我们知道，免疫系统(immune systems)是人类除了神经系统外的第二信号系统。免疫是生物体对外来大分子特别是蛋白质和糖类的一种反应。生物体能够把外来原生质(抗原，antigen)同其自身的原生质区分开来，进而对病原菌、毒素等有害的异物产生抗体(antibody)和中和反应，这是有机体的普遍现象——免疫现象。它的最大特点是免疫记忆特性、抗体的自我识别能力和免疫的多样性。尽管自然界中所有微生物都能作为抗原起作用，而免疫能抵御它们，促进白细胞的噬菌作用。

从信息论的观点来看，免疫系统与神经系统的记忆与识别功能极其类似，利根川进获诺贝尔奖的研究工作——免疫多样性的体细胞发生，初步阐明了人类免疫系统适应外界多样性抗原而形成抗体多样性的机制。Burnet 提出的网络理论初步解释了免疫系统的识别特性。威廉·卡尔文在"脑如何思维"一文中指出"达尔文进化过程在千万年的时间过程中形成新的物种，而在持续数周的免疫反应中产生新的抗体，同样也可以在思维和动作的时间尺度上形成思想"。免疫系统的信号识别、记忆能力与适应环境的免疫多样性，就其结构的复杂程度和处理

信号的功能而言，并不亚于神经网络系统。复杂的自适应具有通常动力学系统所熟悉的性质，包括分层结构、多个吸引盆及许多亚稳态之间的竞争，除此之外，它们还必须有一种能应付并利用环境变化的能力。一种自适应系统的研究方法是构造一个明显的时间层次：一个时间尺度描述真实动力学，另一个较慢的时间尺度考虑非线性方程本身的演化。免疫系统模型、自催化蛋白质网络模型就是在这种思想指导下提出的，主要目的是研究免疫系统的自适应、识别、学习和进化的非线性动力学行为。

生物的免疫系统如图 10.10 所示，由淋巴细胞和抗体分子组成，淋巴细胞又由胸腺产生的 T 细胞和骨髓产生的 B 细胞组成，T 细胞分泌淋巴细胞活素，B 细胞分泌血清抗体，淋巴细胞活素和血清抗体都可与抗原进行专一性结合产生抗体。此外，为维持免疫平衡，抗体间有抑制和促进作用。

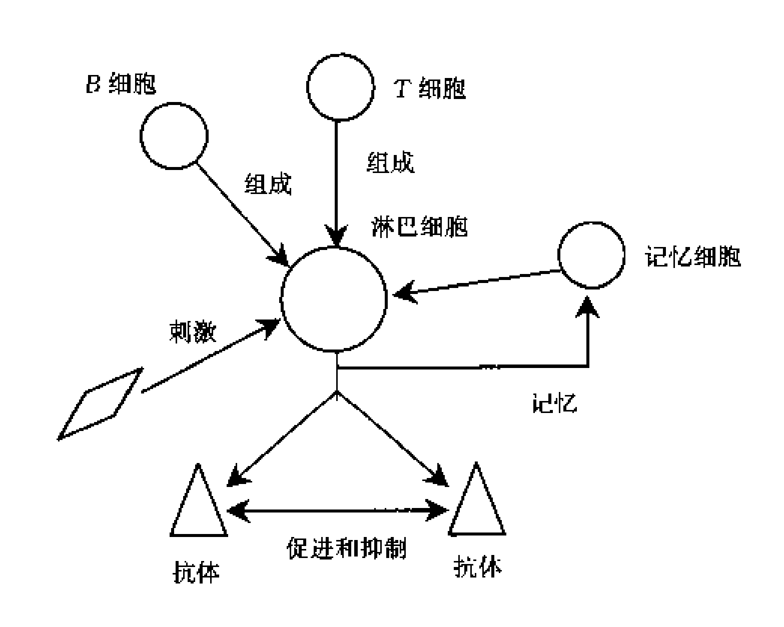


图 10.10　生物免疫机制的抽象模型

生物的免疫系统虽然十分复杂，但其呈现出来的抵御抗原的自适应能力却是十分明显的。免疫系统建模方法包括微分方程模型法、元胞自动机法、分类系统法以及遗传算法。遗传算法将外来侵犯的抗原和免疫系统分别与实际求解问题的目标函数以及问题的解相对应，来模仿生物体免疫机制。这对于改进遗传算法的性能、设计机器学习和人工生命模型是非常有意义的。

从免疫系统的特征上分析，免疫系统通过细胞的分裂和分化作用，产生大量的抗体来抵御各种抗原，这对应于遗传算法中个体的多样性；免疫系统具有维持免疫平衡的机制，通过对抗体的抑制和促进作用，能自我调节产生适当数量的必要抗体，这对应于遗传算法中个体之间基于适应度的选择，利用这一功能可以提高遗传算法的局部搜索能力；免疫系统产生抗体的部分细胞会作为记忆细胞而被保留下来，对于今后入侵的同类抗原，相应的记忆细胞会迅速激发而产生大量的抗体。如果遗传算法中能利用这种抗原记忆识别功能，则可以加快搜索速度，提高遗传算法的总体搜索能力。

美国新墨西哥大学的 S. Forrest 和洛斯·阿拉莫斯实验室的 A. Perelson 首先提出了免疫系统的遗传算法建模方法。首先，系统对输入的抗原进行识别；如果识别的抗原是记忆抗原，则

从记忆细胞中取出相应的抗体组成初始种群，否则随机产生初始种群，计算个体(抗体)的适应度之后执行向记忆细胞的分化；如果抗原是新抗原，用当前种群中适应度高的个体替换记忆细胞中的适应度低的个体，否则将适应度高的个体加入到记忆细胞中去。对种群中个体的选择概率按照抗体的促进和抑制规律确定，由适应度概率和浓度概率两部分组成。新一代的种群产生的选择、交叉和变异操作与一般遗传算法相同。

Farmer, Packad, Perelson 提出的基于免疫原理分类系统模型——FPP 模型。该模型用遗传算法实现免疫系统模型的学习，与 8.2 节介绍的遗传学习分类系统不同的是，需要同时随机产生一定大小的抗原种群，和作为分类系统解的抗体种群。其分类系统评价部分如图 10.11 所示，其步骤如下：

第 1 步　从抗原种群中随机选择一个抗原个体；

第 2 步　从大小为 N 的抗体种群中随机选择 n 个样本抗体；

第 3 步　计算 n 个样本抗体与挑选出来的抗原个体之间的一致程度，该一致程度作为样本抗体的得分，计算方法为：

$$Score = 抗原 \oplus 抗体 \tag{10.8}$$

式中⊕为异或运算。

第 4 步　选择得分最大的抗体，得分相同的场合再从中随机选择。对该抗体的适应度增

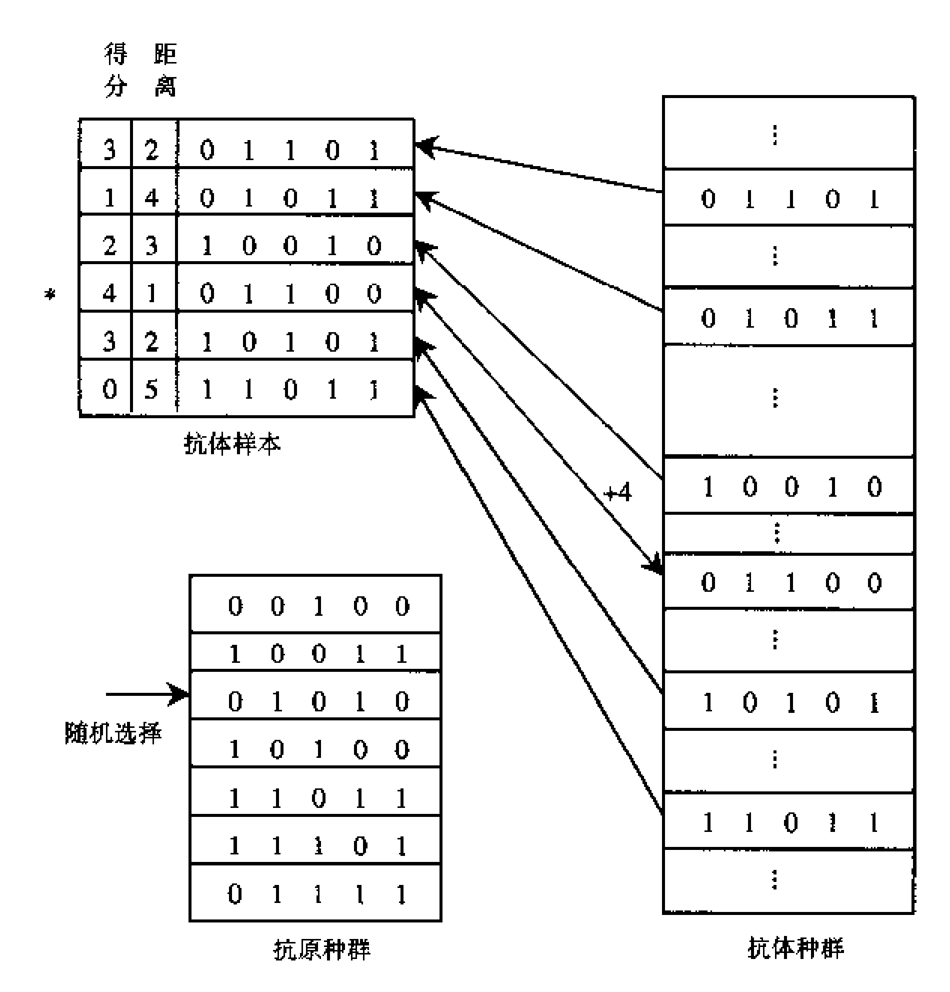


图 10.11　免疫系统模型

加得分值。

上述过程重复一定次数，即可完成对抗体种群的评价，即分类系统的评价。这种基于免疫原理的评价方法体现了免疫多样性的优点，通过累加抗体与抗原一致程度的得分来评价抗体，抗体的适应度值不接受来自于该抗体无关系抗原的直接影响，这样随着多样性的抗原个体被选择，很容易产生适应度高且多样性的抗体。

目前，有关模拟免疫系统方面正在进行的研究已跨越了生物学和医学的疆域，引向一个新的领域——具备免疫特征和功能的人工生命系统。除了上述研究之外，基于免疫网络理论设计自治式多Agent系统、免疫型自适应系统、免疫型安全系统或抗干扰系统以及面向医学应用的数字免疫监控系统等研究，可望在不久的将来得到实际的应用。

10.5　进化硬件问题

进化硬件(Evoluable Hardware, EHW)的概念是1992年由日本ATR的H. deGaris提出的。其目的在于研制出一种自治(autonomous)工作方式的硬件，能够像生物一样根据环境的变化而改变自身的结构以获得最优的适应性能。它是一种进化算法与可编程逻辑器件相结合的新的硬件设计方法。以前的硬件在设计和制造后，其结构几乎很难变更，性能的提高主要依赖控制和管理手段的改进。

近年来，可编程逻辑器件(Programmable Logical Device, PLD)获得了迅速的发展，特别是现场可编程门阵列(Field Programmable Gate Array, FPGA)在器件的选择和内联上提供了更多的自由度，用户可通过编程实现专用集成电路(ASIC)，达到改变硬件功能的需要。但FPGA自身没有改变的能力，用进化算法的方法实现这种改变，能够更好地迎合用户的高级需要，如容故障性、自我修复、多用途以及具备一定的学习能力等。

进化硬件的研究虽然只有短短几年时间，但已经取得的一些初级应用成果，赢得了学界和工业界的重视。以进化硬件为主要内容的国际会议International Conference on Evoluable Systems, ICES迄今为止已举办了三届(1999年在埃及)，IEEE Trans. on Evolutionary Computation在1999年9月组织了"进化硬件特辑"，讨论这方面的进展。按硬件类型不同，EHW可以分为数字型进化硬件和模拟型进化硬件两大类，数字型进化硬件又可以进一步分为进化计算型和Bio－Inspired型，其中以进化计算型的研究和应用最多。模拟进化硬件着眼于用进化算法自动生成模拟电路或进化LSI。

本节将介绍进化硬件的一些基本问题和几个不同类型的实例。

10.5.1　进化硬件的基本思想

可编程逻辑器件PLD或FPGA的主要优点在于，其结构是由结构位串(architect bits)确定的，而且是可重配置的，不同的结构位串决定了可编程器件的不同结构。根据硬件所要实现的功能，产生相应的结构位串，下载(download)到可编程器件上。因此，当硬件所处的环境发生变化时，硬件的结构也需要做相应的调整，就可以用这种改变结构位串的方法来实现。图10.12为FPGA的基本结构示意图，FPGA由布线资源围绕的可编程功能块(functional block)构成阵列，又由可编程I/O单元围绕阵列构成整个芯片。排成阵列的功能块由布线通道中的可编程内连线来实现一定的逻辑功能。每个功能块有不同的逻辑功能，可以表示为特定的结

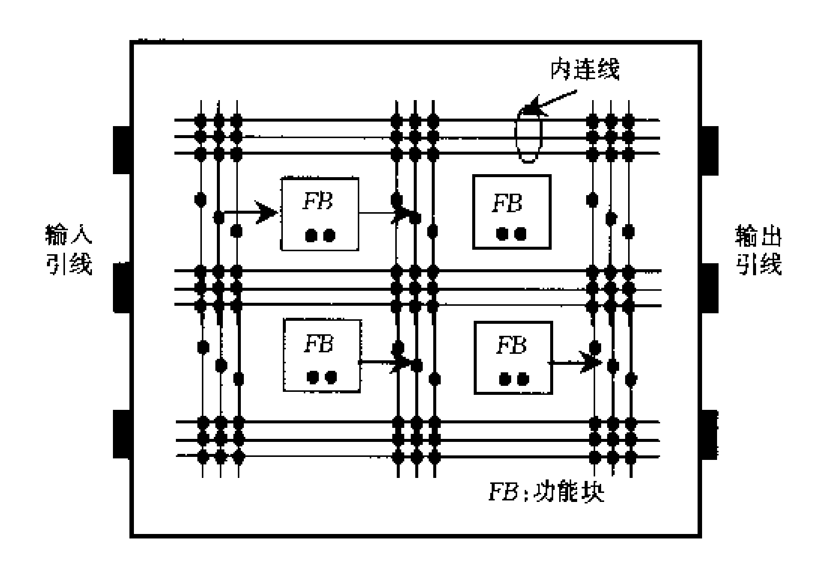


图 10.12　FPGA 的结构示意图

构位串。内连线的连接结点(图中的实心结点)的开关状态在器件被制造后可以再被加载和修改,因此可以作为可重置的二进制结构位串,这个结构位串就能够决定器件的功能。与 FPGA 不同的是,PLD 通常是通过修改具有内连电路的逻辑功能来编程的。FPGA 比 PLD 具有更高的集成度,更复杂的布线结构和逻辑实现,更快的重构速度,因而更适合选择作为进化硬件的对象。

进化硬件正是基于可编程逻辑器件的结构可重配置的特点,将结构位串看作进化算法中的染色体,所要实现的功能作为评价函数,借助进化算法来实现硬件的设计,如图 10.13 所示。将 FPGA 的结构位串当作遗传算法的染色体,用遗传算法去寻找更好的硬件结构。一旦发现了足够好的染色体,则将它置载在 FPGA 上。

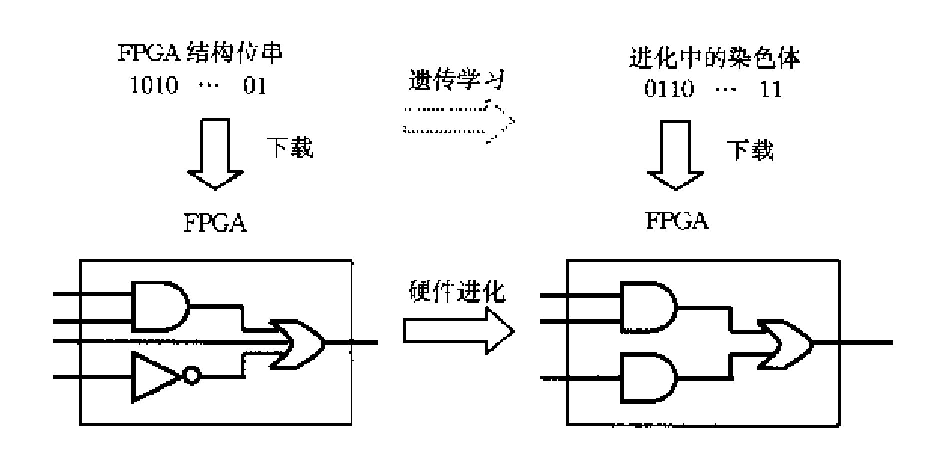


图 10.13　EHW 基本原理

H. deGaris 认为 EHW 的进化有两种方式：一种是外部(extrinsic)进化,也称为离线(off-line)进化;另一种是内部(intrinsic)进化,也称在线进化。通常 PLD 的进化方式为外部进化,通过硬件描述语言(HDL)来描述电路,借助软件把这种语言转换成电路相应的结构位串。这也是目前数字型硬件主要的进化方式,因为结构位串的长度很长,遗传算法求解过程耗时长。在线方式需要解决结构规模、功能复杂性与进化速度的矛盾,Kajitani 等提出了用变长染色体

遗传算法(如 messy GA)来解决这一问题。但从实用的角度考虑,目前还存在许多问题。

10.5.2　数字型进化硬件实例

最近,日本电子技术综合研究所与 NEC 合作研究的数字型进化 LSI,将 PLD 部、遗传算法实现部以及作为外部接口的 CPU 集成在一块芯片上,并用它来开发假手和自治式移动机器人。图 10.14 所示的是为残疾人开发的筋力自动控制的假手。当人的手臂肌肉运动时,皮肤表面会产生一定的电位,不同的动作如握紧、松开等,筋力的模式各有不同。用进化硬件的方法自动识别不同模式的筋力,即在 EHW 上实现这种模式识别电路,从而实现控制假手动作的目的。先前开发的控制型假手用多个系列适应不同患者的需要,通常患者需要一个月左右的时间才能比较适应,用 EHW 方法开发的假手能够在发生伤残后立即使用,很快适应患者的肌肉萎缩变化。该产品已在 1998 年商品化。



图 10.14　筋力自动控制的假手

图 10.15 所示的是基于 EHW 方法开发的自治式移动机器人"Evolver",用于追踪滚动的小球。机器人用多个距离传感器跟踪小球时,若其中某个传感器被损坏,机器人仍然能够根据剩余传感器的信号快速重构控制电路,并且重构后机器人的适应时间很短。



图 10.15　自治移动机器人(Evolver)



图 10.16　元胞自动机的进化硬件板(Firefly)

瑞士联邦工学院制作的萤火虫(Firefly)进化硬件板,如图 10.16 所示,属于一种 Bio－Inspired 型进化。该系统是一个进化的一维非均匀元胞自动机,56 个元胞中每一个都包含一个基因,用基因表达元胞状态转移的规则,这些基因都是从随机初始状态开始进化的,板上包含下述元件:

① LED 元胞状态显示器(最上面一排);

② 人工置入元胞初始状态的开关(LED 下面一排);

③ Xilinx FPGA 芯片(开关下面一排);

④ 底左部是显示器和控制元胞程序设计计算的"时间步"和"配置"参数按钮;

⑤ 中左部是一个同步指示器;

⑥ 中底部为带有手动可调频的主时钟脉冲发生器;

⑦ 底右部为一个状态规则表的 LED 显示器和进化时得到的适应度值;

⑧ 最左部为一个电源电缆。

元胞自动机是 Von Neumann 提出的一种具有自繁殖和自修复能力的机器模型。目前,日本 ATR 的 CAM 计划也正在进行与 Firefly 类似的研究。

10.5.3 模拟型进化硬件实例

模拟型进化硬件用进化算法自动生成模拟电路或进化 LSI。这方面的代表性研究来自于 Stanford 大学的 Koza。他用遗传程序设计的方法,进化模拟电路的构造,自动获取电路的结构及其元件(R, L, C 等)的特征值,其应用实例包括交叉(crossover)调制滤波器、非对称 butterworth 滤波器、运算放大器等。图 10.17 所示的是日本旭化成微系统株式会社正在开发的用于移动电话的中频滤波器,使用了模拟进化 LSI 的方法。产业界对这种模拟型进化硬件寄予了很高的期望。

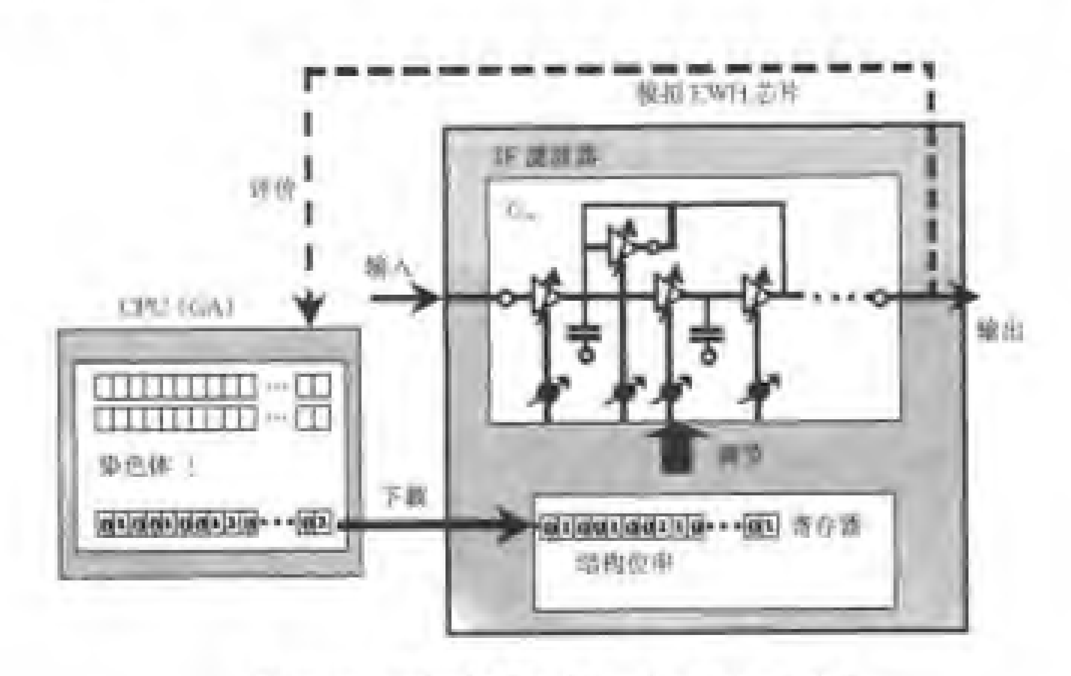


图 10.17 中频滤波器的模拟进化芯片结构示意图

进化硬件作为一个新的研究方向,其优越性在于:我们不需要对硬件的结构有很深入的了解,只需要根据其外部特性就可以用进化算法的手段自动寻优。但进化算法的寻优速度因具体问题不同差异很大,目前迫切需要研究高速的进化算法实现手段。此外,进化硬件的理论分析也是一个重要方面,有的问题如模式识别问题,凭借对系统外部特性的不完备测试,很难保证系统的可靠性。总之,目前进化硬件的困难还很多,距离较广泛的应用还有很长的路程。

著名科学家霍金在新千年前夕回答记者提问时认为,在未来 100 年,我们也许会发现一个关于宇宙基本规律的完整理论,但在这些规律下,我们可以建立的生物复杂性和电子系统复杂性却是无限的。英国雷丁大学控制论专家凯文·渥维克(K. Warwich)教授正在从事一项有很

大风险的研究工作，试图将一个与人体神经系统连接的"通心芯片"置入人体内，"通心芯片"能够在两人之间传送情感、痛苦和生理运动时的神经冲动。一旦这项研究成功，将为残疾人提供一种向健全人学习提高四肢控制能力所需要的神经冲动的方法。今后，人类也可以直接依靠思想来彼此交流。

进化机器的工作会不会像克隆人因可能的滥用而受到限制？或者机器发展到超过人类的智商和能力，人类难以控制它的地步呢？应该说，这样的担心不是多余的。在物理上，电子系统的复杂性和速度有一个极限，从计算机的速度而言，光速是一个实际的界限，人们可以把电路变小来改进速度，但最终有一个受物质之原子属性的约束极限，实际上当硅集成块和元件间的距离接近 10 nm 时，电子从邻近元件逸入的概率很有限，便产生了"隧道效应"的现象，它是高集成电路块工作不可靠的原因之一。如果将生物系统的复杂性有效地引入到电子系统中(如 ATR 的人工脑)，机器的未来是难以估量的。

10.6 进化对策论

10.6.1 "囚犯困境"问题

两个囚犯被关在一个隔离的牢房里，彼此不能交流。两个囚犯面临两种选择：合作(cooperate，简记为 C)或者背叛(defect，简记为 D)。如果只有一个囚犯选择背叛，该囚犯就会得到奖励，另一个囚犯则受到惩罚。如果两个都选择背叛，则两个都会被受到折磨。如果两个都选择合作，则两个都会受到中等的收益。这样，不论另一个囚犯如何选择，选择背叛总比选择合作产生更高的收益。但是如果两个都选择背叛，其收益比两个都选择合作得到的收益少。这就是对策论(game theory)中著名的"囚犯困境"(the prisoner's dilemma)问题，它是 1952 年美国兰德公司 M. Flood 和 M. Dresher 给出的。图 10.18 列出了该问题的对策收益表。表中存

		囚犯 2	
		合作 C	背叛 D
囚犯 1	合作 C	(r_1, r_1) (3,3)	(r_2, r_3) (0,5)
	背叛 D	(r_3, r_2) (5,0)	(r_4, r_4) (1,1)

图 10.18 "囚犯困境"问题对策收益表

在下列约束：

$$2r_1 > r_2 + r_3 \tag{10.10}$$

$$r_3 > r_1 > r_4 > r_2 \tag{10.11}$$

在对策论中，我们通常将对策的一方称为局中人；将局中人每步对策选择的所有顺序组合

称为一个策略。每个局人总是希望在竞争过程中取得尽可能好的收益,因此需要在策略集合中选择好的策略对付自己的对手。在上述对策游戏中,第一个约束的含义为:始终选择相互合作的策略(CCCCCC…)其收益比轮流选择合作和背叛的策略(CDCDCDCD…)收益大。第二个约束的含义是:根据 Nash 均衡论,只有(C,C)位于 pareto 最优前沿,选择合作的双方有机会达成所谓的"双赢"。但实际上对局双方才难以达成这样的默契,只有选择背叛的动机才会使自己处于有利的地位。

10.6.2 反复的"囚犯困境"

如果重复上述的对局游戏,问题要复杂得多,我们称之为"反复的囚犯困境"(the iterated prisoner's dilemma)。现实世界中这样的例子不胜枚举,例如,鸟类有互相帮助对方除去羽毛上扁虱的行为,就是在表演这种"反复的囚犯困境"游戏。对于每只鸟而言,啄出自己身上的扁虱是重要的,但是它无法处理自己的头部,所以只好让自己的同伴帮助处理,只有它在日后回报这份恩情才显得公平。虽然这项服务花费的时间和精力不多,但是假设有一只鸟作弊,专门请别的鸟除虱却不进行回报,那么它不必付出代价就得到所有好处。列出各种情形,不难发现这正是一个囚犯的困境。互相合作(啄出对方的扁虱)是非常好的情形,但是还有另一种试探会更好,就是拒绝回报对方。互相背叛(拒绝啄出对方的扁虱)是相当糟的情形,但还有更糟的情形,就是费尽力气替别的鸟啄出对方的扁虱,而自己的头上积满了虱子。

"反复的囚犯困境"提供了足够多的策略领域,不像简单的一步对局,很容易就可预期背叛是唯一合理的策略。如果局中每步选择都是独立的,不受以前的双方对策结果的影响,始终背叛同样是合理的策略。问题在于局中人的动机不是单一的,只要对它有利(自私的基因驱动),它就会根据以前的对策结果来判断是选择进一步合作还是背叛。假定一个局中人有这样的复合动机:若对手选择持续的合作则"投桃报李"(return the compliment)选择持续的合作,若对手选择持续的背叛则"针锋相对"(tit for tat)也选择持续的背叛。面对这样的局中人,尽管背叛的利益诱惑相对于合作要大,只要选择步数足够多,选择始终背叛的策略是不明智的。当然,其他的复合动机也是大量存在的。由不同的复合动机会产生的不同竞争合作行为模式和结果,从中寻求一些合理的行为模式成为研究"囚犯困境"问题的目的所在。由于该问题的复杂性,并且可以推广到两个以上局中人的场合,从而能够体现深刻的现实世界背景,成为模拟社会、经济以及军事系统中竞争和合作行为机制的经典模型。

美国政治社会学家 Robdr Axelrod 是研究"囚犯困境"问题的著名学者,他于 20 世纪 80 年代初向一些对策论专家发出征集"囚犯困境"策略的请求,然后从中选出了 14 个策略,再加上作为对比的一个随机策略,总共 15 个策略。用计算机处理有 15×15=225 次对局的联赛(round robin tournament)结果。每次对局有 200 步,每个策略计算一次对局的累积分及 15 次对局的平均累积分,然后从中挑选出最佳的策略。发现多数策略的平均累积分不超过 600 分(理论上策略的最大累积分为 1 000),只有一个被称为"针锋相对"(tit for tat)的策略是赢家,平均累积分在 600 以上,这个策略表面看来很简单,首先选择合作,然后始终选择对手的前一步行动作为自己的应对。后来陆续有人向 Axelrod 建议了很多其他的策略,与"针锋相对"的策略比高。例如,一种被称为"天真的试探者",它与"针锋相对"基本类似,只是每隔固定的步数自作主张随机地决定背叛;一种被称为"自责的试探者",它与"天真的试探者"基本类似,所不同的是它会允许对手有一次免遭报复的背叛,从而中止反复互责走向合作。Axelrod 收到的策略中有的诡诈、有的善良,如超级宽恕"两怨还一报":允许对手背叛两次才报复一次。Axel-

rod 组织了第二次扩大的联赛(63 个策略),结果发现,它们中有的策略可能从“针锋相对”一方得到好处,但究其平均表现都不如“针锋相对”,而“两怨还一报”表现仅随其次。Axelrod 认为一个好的策略应该具备善良、宽恕、理解的品质。所谓善良是指不首先背叛,宽恕是指容忍一定程度的背叛,理解包括既能理解对方的策略也能被对方理解两个方面。他的研究工作发表在 1981 年美国《科学》期刊上,获得了美国科学促进协会的 Newcomb Cleveland 奖。

Axelrod 的第一次联赛中的策略大约一半是善良型的,“针锋相对”在这种生态环境下取得了胜利。第二次联赛中虽然增加了规模,“针锋相对”还取得了胜利,这说明“针锋相对”是一种健壮的策略。但如果参赛的策略都是诡诈型或善良型的,那么“针锋相对”还能赢得胜利吗?答案是否定的,因为有些策略对它极其不利。“两怨还一报”虽然没有参加第一次联赛,如果参加的话,它就会是赢家。这说明“针锋相对”还不够稳定。后来 Robert Boyd 和 Jeffrey Lorberbaum 提出了“两怨还一报”与一种多疑的“针锋相对”的混合策略,表现出了更好的稳定性。

10.6.3　用遗传算法进化反复的“囚犯困境”的策略

在生物进化论的研究中,生物学家 J. M. Smith 提出了“生物进化的稳定策略”(Evolutionarily Stable Strategy, ESS)的概念,认为生物种群的大部分成员都采用某一策略,而这种策略又优于其他策略,这种策略就是生物进化的稳定策略。这一概念与上面提到的合理行为模式含义相近,对于大多数生物个体来说,最好的策略也许是随着生物种群的大多数成员在做什么就做什么。由于生物种群的其他成员也在力图最大限度地扩大自身的利益和成就,因而生物能够持续存在的一种策略,就是它一旦形成,任何行为异常的生物个体的策略都不可能与之比拟的策略,偏离 ESS 的行为将受到自然选择的惩罚。

基于这一思想,Axelrod 将遗传算法方法应用于“囚犯困境”问题上,用于发现该问题中的好策略。

Axelrod 给出的学习囚犯困境策略的遗传算法中,将局中人的策略表达为个体的基因编码,每个基因码为 D 或 C。

为简化处理,考虑确定性策略并使用前三步选择当前步。每一步共有四种可能的选择结果,即总共有 $4^3=64$ 种不同的前三步记录。这样需要产生 64 个基因码来表示面对前三步记录时的选择。其编码方法很简单,直接随机产生 64 个基因码 D 或 C。其解码方法较为特别。假定四种对局选择结果分别用四进制的四个数表示为:

$$(C,C)=0,\quad (D,C)=1,\quad (C,D)=2,\quad (D,D)=3$$

则一种前三步记录可以联合记为一个 3 位四进制数,其十进制数值表示该记录指向基因码的位置,例如:

- $(C,C)(C,C)(C,C)=(000)_4=(0)_{10}$,若第 1 位的基因码为 C,则表示$(C,C)(C,C)(C,C)\to C$,即三次合作后继续选择合作。
- $(C,D)(D,C)(C,C)=(210)_4=(36)_{10}$,若第 37 位的基因码为 C,则表示$(C,D)(D,C)(C,C)\to C$,即当合作恢复后继续合作。
- $(D,D)(D,D)(D,D)=(333)_4=(63)_{10}$,若第 64 位的基因码为 D,则表示$(D,D)(D,D)(D,D)\to D$,即三次背叛后继续背叛。

余此类推。任意一种前三步记录在染色体中都有一种当前步选择,这就形成了一个策略描述。局中人在三步之后的每一步选择都可在 1~64 位基因码中检索。

为了获得对局开始时的策略,需要假设前三步的选择,要求另外 6 个基因码。这样表达局

中人的策略对应的染色体基因总共需要 70 位，整个策略空间大小有 $2^{70}\approx10^{21}$ 之大。

策略遗传学习的算法分以下几个步骤：

第 1 步　随机地产生初始种群，种群大小为 20，一个个体代表一个策略。

第 2 步　评价每个策略收益，局中人采用该策略，对手随机挑选的策略进行对局，对局步数为 151，计算局中人的得分。共随机安排 8 次这样的随机对局，获得局中人的平均得分即为策略的评价值。

第 3 步　基于策略的评价值选择一定比例的策略，用于产生新一代的个体策略。

第 4 步　对选择出来的个体实行交叉和变异操作，生成新一代的种群。

第 5 步　若未达到预定世代数(如 1 000 代)，则返回到第 2 步。

从上述算法的实算结果发现，下列行为模式包含在末代的大多数个体中：

① 稳定合作模式：经过 3 次连续合作后继续合作，即 $(C,C)(C,C)(C,C)$ 后选择 C。

② 惩罚背叛模式：对手放弃合作选择背叛时则背叛，即 $(C,C)(C,C)(C,D)$ 后选择 D。

③ 不记前嫌模式：当合作恢复后继续合作，即 $(C,D)(D,C)(C,C)$ 后选择 C。

④ 加强合作模式：合作恢复后继续保持，即 $(D,C)(C,C)(C,C)$ 后选择 C。

⑤ 接受教训模式：三次背叛后背叛，即 $(D,D)(D,D)(D,D)$ 后选择 D。

现实世界中的“囚犯困境”问题是否这样简单呢？首先，策略的选择不局限于由前三步决定当前步，需要建立一个统一的信念修正模型来包容一个完备的策略集。现实世界中存在很多处于“合作”和“背叛”之间的中间选择，对局双方的力量随时间的演变各有消长，不同选择的利益也会有所变化。这时就有必要从动力学的角度考虑复杂的策略描述和建模问题，希望稳定的策略从所建立的模型突现出来的。例如，可用神经网络来模拟局中人的策略选择，用遗传算法或进化规划等方法来训练神经网络。

应该说明的是，“囚犯困境”问题是对策论中一种非零和游戏。一般而言，用进化算法研究对策论中稳定策略问题，被称为进化对策论(Evolutionary Game Theory)。它可以作为人工生命研究方法之一，用于模拟社会、经济、军事、安全领域内复杂的行为模式，例如核战略、证券交易、信贷评价、企业竞争和合作、网络安全等系统中的 Agent，若借助系统分析和决策方法，即便我们能够抽象、演绎、归纳出系统近似的结构和模型，一般很难反映 Agent 的意愿、偏好、行为效用的不可预知性，系统发展的轨迹会因为没有把握的行为模式发生偏离。系统学与动力学相结合、计算方法与算计方法相结合模拟生态系统中竞争和合作行为的突现规律，是解决类似问题的新的方法论。这方面的研究进一步拓展了人工生命、遗传算法的应用领域，同时也是具有应用潜力一个方向。

参考文献

[1] Langton C G. Artificial Life. In: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems. Los Alamos, New Mexico, Addison-Wesley, 1989

[2] Langton C G et al(ed). Artificial Life II. In: Proceedings of the Workshop on Artificial Life, Sante Fe, New Mexico, Addison-Wesley, 1992

[3] Mitchell M. Computer Models of Complex Adaptive Systems. Santa Fe Institute, New Scientist, 1993(2): 10~15

[4] Hopfield J J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In: Proceedings of the National Academy of Sciences USA 79, 1982, 2554～2558

[5] Scott S D et al. HGA: A Hardware－Based Genetic Algorithm, In: Proceedings of the 1995 ACM/SIGDA Third Int. Symosium on Field－Programmable Gate Arrays, 1995, 53～59

[6] Fogarty T G, Bull L, Carse B. Evolving Multi－Agent System. In: Genetic Algorithms in Engineering and Computer Science, Winter G(ed). Wiley, 1995, 3～22

[7] Hillebrand E, Stender J. Many－Agent Simulation and Artificial Life. IOS Press, 1994

[8] Sian S S. Extending Learning to Multiple Agents: Issues and a Model for Multi－Agent Machine Learning. In: Machine－Learning－EWSL－91, Y. Kodratoff(ed). LNCS 482, 1991

[9] Axelrod R. The evolution of Strategies in the Iterated Prisoner's Dilemma. In: Genetic algorithms and Simulated Annealing, Davis L D(ed), Morgan Kaufmann, 1987

[10] Collins R J. Studies in Artificial Evolution. Doctoral Dissertation, Artificial Life Laboratory, Department of Computer Science, University of California, 1992

[11] Kawata M, Toquenaga Y. From Artificial Individual to Global Patterns, TREE 9(11), 1994, 417～421

[12] Angeline P J. Evolutionary Algorithms and Emergent Intelligence. Doctoral Dissertation, the Ohio State University, 1993

[13] Kitamura S. System Theory of Function Emergence. In: Proceedings of Symposium on Emergent System, Tokyo Institute of Technology, Japan, 1996, 1～2

[14] Hoshino T. Summary of Research on Artificial Life Systems. In: Proceedings of Symposium on Emergent System, Tokyo Institute of Technology, Japan, 1996, 35～36

[15] Hoshino T. Evolution of Systems by Artificial Life Technique. In: Proceedings of Symposium on Emergent System, Tokyo Institute of Technology, Japan, 1996, 37～38

[16] Sannomiya N. A Study on Emergent Phenomena in Fish Behavior. In: Proceedings of Symposium on Emergent System, Tokyo Institute of Technology, Japan, 1996, 51～54

[17] Sergio P, Silverra P, Massad E. Modeling and Simulating Morphological Evolution in an Artificial Life Environment. Computers and Biomedical Research 31, 1999, 1～17

[18] Mitchell M, Forrest S. Genetic Algorithms and Artificial Life. Santa Fe Institute Working paper 93－11－072, 1993

[19] Schuster P. How does Complexity Arise in Evolution ? Santa Fe Institute Working paper 95－10－012, 1995

[20] Fogel L J. Top－Down Evolutionary Engineering. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 11～16

[21] Casti J L. Emergent Phenomena and Computer Worlds, In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 1～10

[22] Makita Y, Hagiwara M. Knowledge Extraction Using Neural Network by an Artificial Life Approach. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 177～186

[23] Nerome M, Yamda K. Competitive Co－evolution Model on the Acquisition of Game Strategy. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 224～232

[24] 中野馨. 生命をつくる. Computer Today, 1999(92): 38～43

[25] 石田好輝. 復雑系としての免疫システム—免疫系に学んだ情報システムにだむけて. システム/制御/情報, 1998, 42(9): 487～494

[26] A. K. デコードニー. 進化のシミュレーション：虫ガクテリアを食べるよろになるまで. サイエンス, 1989-7, 1989
[27] 特集, 復雑系とコンピコータ. Computer Today, 1997(81)
[28] 創発システム公開シンポジウム資料, 文部省科学研究費補助金重点領域研究, 創発的機能形成のシステム理論. 東京工業大学, 1997. 12
[29] 和田健之介. 遺伝的アルゴリズムと機械の進化. 数理科学, 1990(328)
[30] 通口哲也. 進化型ハードウエア. IPSJ Magazine, 1999, 40(8): 795～800
[31] 刘健勤. 人工生命理论及其应用. 北京: 冶金工业出版社, 1997
[32] (美)米歇尔·沃尔德罗. 复杂-诞生于秩序与混沌边缘的科学. 生活·读书·新知 三联书店, 1998
[33 郭凯声 等. 数学游戏(上、下). 北京: 科学技术文献出版社, 1999
[34] 李国杰. 世纪电脑. 北京: 科学技术文献出版社, 1999
[35] 戴汝为. 组织管理的途径与复杂性探讨. 科学(双月刊), 1998, 50(6): 8～12
[36] 李夏, 戴汝为. 系统科学与复杂性(I). 自动化学报, 1998, 24(2): 200～206
[37] 李夏, 戴汝为. 系统科学与复杂性(II). 自动化学报, 1998, 24(2): 476～483
[38] 托马斯 L C. 对策论及其应用. 靳敏, 王辉青 译. 北京: 解放军出版社, 1988
[39] 李夏, 戴汝为. 突现(emergence)——系统研究的新概念. 控制与决策, 1999, 14(3): 97～101
[40] 吴建兵, 杨杰, 吴月华. 人工生命与人工智能. 模式识别与人工智能, 1998, 9(3): 274～279
[41] 魏洁敏, 刘健勤. 创发 DNA 计算模型及其混沌系统辨识过程. 中南工业大学学报, 1998, 29(3): 494～496
[42] 康立山, 何巍, 陈毓屏. 用函数型可编程器件实现演化硬件. 计算机学报, 1999, 22(7): 781～784
[43] 王煦法, 张显俊, 曹先彬. 一种基于免疫原理的遗传算法. 小型微型计算机系统, 1999, 20(2): 117～120
[44] 张纪会, 徐心和. 一种新的进化算法——蚁群算法. 系统工程理论与实践, 1999(3): 84～87
[45] 欧阳曙光, 贺福初. 生物信息学: 生物实验数据与计算技术结合的新领域. 科学通报, 1999, 44(14): 1457～1468
[46] 成思危 主编. 复杂性科学探索. 民主与建设出版社, 1999
[47] 中国科学院《复杂性研究》编委会. 复杂性研究. 北京: 科学出版社, 1993

第 11 章　遗传算法与图像处理模式识别

图像处理和模式识别是计算机视觉中的一个重要研究领域。在图像处理过程中，如扫描、特征提取、图像分割等不可避免地会产生一些误差，这些误差会影响到图像处理和识别的效果。如何使这些误差最小是使计算机视觉达到实用化的重要要求。遗传算法在图像处理中的优化计算方面是完全能胜任的，目前已在图像校准、图像分割、几何形状识别、图像压缩、三维重建优化以及图像检索等方面得到了应用。本章将介绍遗传算法在图像歪斜校准、图像分割以及基元识别与提取三个方面的应用，这些应用的方法各具特色，可以作为进一步推广和发展的基础。

11.1　图像歪斜校准

在医学图像、遥感图像等处理中，存在图像歪斜校准的问题。例如，根据患者新的 X 光相片 A 与过去拍摄的相片 B 相比较，判断局部小的变化时，为达到一定精确度，相片位置和角度的校准工作是必须的。由于歪斜的非线性性质，而且相片中没有患者身体基准点的标记，用一般图像处理的方法实行歪斜图像校准是很困难的。这项工作通常主要地依靠医师的经验手工进行。这里介绍一种基于遗传算法的图像校准函数辨识方法。

假设灰度图像 A 上一点(x, y)的灰度为 $A(x, y)$。如图 11.1 所示，定义下列简单的非线性变换：

$$
\begin{aligned}
x'(x, y) &= a_0 + a_1x + a_2y + a_3xy \\
y'(x, y) &= b_0 + b_1x + b_2y + b_3xy
\end{aligned}
\tag{11.1}
$$

经过以上变换，得到图像 A'。问题是确定系数 a_0, a_1, a_2, a_3 和 b_0, b_1, b_2, b_3 使图像 A'

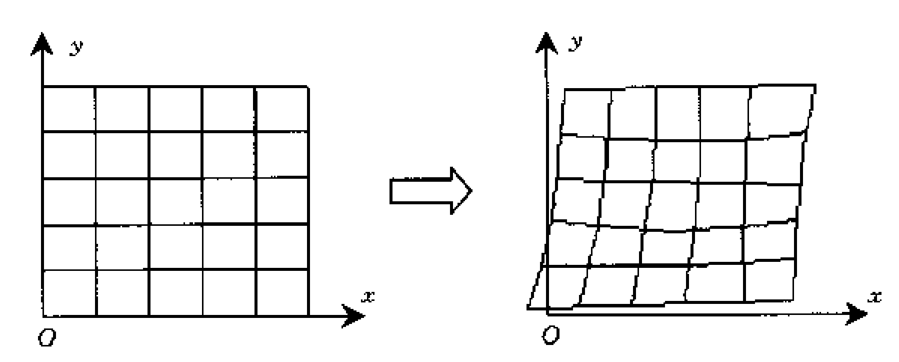


图 11.1　坐标变换

与歪斜图像B之间的误差最小，则我们根据获得的变换图像推断歪斜图像B中发生了变化的部分。遗传算法应用于变换函数的辨识，图11.2给出了这种方法的过程。考虑对系数(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3)进行个体染色体编码，个体的适应度可根据其系数计算变换后图像A'与歪斜图像B之间的误差进行评价，误差值可按式11.2计算。个体的误差值越小，则其适应度越大。

$$\sum_{x'}\sum_{y'}(A'(x',y') - B(x',y'))^2 \tag{11.2}$$

应该特别指出的是，由于未考虑歪斜图像灰度的变化(除局部的变化外)，在对于歪斜之外的变化很大的场合，用这种方法进行图像校准是不适合的。

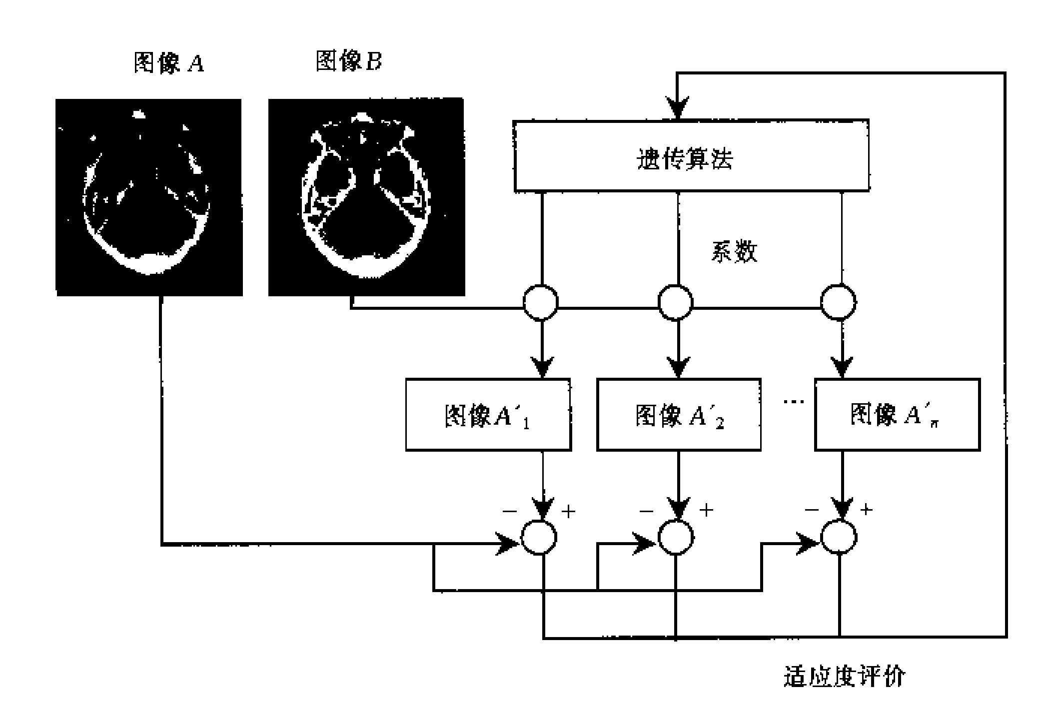


图11.2 基于遗传算法的图像歪斜校准

11.2 图像分割

图像分割是图像处理和前期视觉中的基本技术，是大多数图像分析和视觉系统的重要组成部分。图像分割是应用一种或多种运算将图像分成一些具有类似特性(如颜色、纹理、密度等)的区域，主要有阈值方法(thresholding)和区域方法两大类。前者利用灰度频率对分布信息进行分割，一般可分为直方图法、最大类间方差法、最小误差和均匀误差法、简单统计法、概率松弛法、FCM(fuzzy c－means)模糊聚类算法、马尔可夫随机场法、神经网络方法等10种算法。后者利用局部空间信息进行分割，将具有相似特性的像素集合起来构成区域，主要有区域生长法(region growing)和分裂合并(split－and－merge)法。

阈值方法因其简单且性能稳定而成为图像分割中的基本技术。但在许多情况下，一幅图像中的物体可能由于表面的灰度和颜色不同而在灰度级上有不同的反映，简单的二值分割不能反映图像的特点，而多阈值分割将能为后续处理提供更多的信息。传统的各种阈值方法的

提出，首先是基于单个阈值的，但它们不难推广到多阈值的图像分割中；然而在寻求最佳的多阈值时，传统的算法是用穷尽的搜索方法寻求最优解，因此需要大量的计算时间，从而限制了它的使用。基于模糊聚类的阈值选择作为一种图像的软分割方法，能够处理图像局部区域边界不清的场合。它基于目标特征(如距离、矢量或熵等)分离目标，为每个阈值按相对于每一类聚类中心的距离指定一个隶属度，用最大隶属度法将获取的模糊阈值去模糊，作为分割阈值。

在区域方法中，区域生长法由于生长准则的选取不仅依赖于具体问题本身，也与所用的图像数据有关。若不考虑像素间的连通性和邻近性，会出现无意义的分类结果。分裂合并法被认为是很有希望的一种分割方法，它先人为地将图像划分为若干个规则区域，以后按性质相似的准则，反复分开特性不一致的区域、合并具有一致特性的相邻区域，直至形成一张区域图。这种方法能充分组合图像的全局和局部信息，采用图像四叉树的表达方法便于建立层次数据结构。但也存在区域初始划分和选择区域性质一致性度量、边界模糊性度量两个重要的问题。

本节将结合传统的图像分割技术，讨论遗传算法应用于图像分割的几个典型思路和方法。

11.2.1　遗传学习的分类系统方法

1991 年，Bhanu 提出了一种基于遗传学习的分类系统方法应用于图像分割。如图 11.3 所示，该分类系统规则表有 100 条分类规则，规则表示为 IF－THEN 形式。规则的前件有 50 个参数组成，其中包括 4 种颜色(灰、红、绿、蓝)的特征值，每种颜色有 12 个特征值，还有两个户外特征值(即摄像时刻和天气情况)；规则的后件是用于分类的两个分割参数(也可以更多)。每条规则重要程度用一[0,1]区间内的数值评价，这样规则表中有 100 个分类器。将遗传学习的方法用于图像分割的学习算法可以描述如下：

第 1 步 (特征值获取)　输入需要分割处理的图像，计算其 50 个特征值。

第 2 步 (分类器的适应度计算)　分别求出处理对象前件部与 100 个分类器的前件部的匹配程度，该匹配程度度用距离测度计算，作为各个分类器的适应度。

第 3 步 (分割评价)　取出适应度高的 10 个分类器，将其后件中的分割参数应用于输入图像的分割，计算分割优劣程度的评价值，即规则的分割评价值。

第 4 步 (交叉、变异操作)　对规则的后件分割参数进行交叉和变异操作，产生新的规则后件。

第 5 步 (分割再评价)　将新产生的规则后件中的分割参数再次应用于输入图像的分割，计算出分割优劣程度的评价值，得到规则的新分割评价值。

第 6 步 (分割再评价)　在 10 条新产生的规则后件中，选择分割评价值最高者作为新分类器的后件，并将输入图像的 50 个特征参数作为新分类器的前件，用这个新分类器取代规则表中的最差者。

上述学习算法根据规则的前件部中图像特征参数的学习，用遗传算法生成用于图像分割的控制参数。这里的遗传学习属于密歇根方法，遗传算法中的个体从一幅图像信息中抽取生成，因此一次的学习过程只涉及一幅图像的评价。重复这样的学习过程，可以获得一个不断进化的图像分割系统，有助于提高图像分割的性能。

对于图像分割系统的性能而言，分割评价指标的选取十分重要。不仅要考虑分割出的目标(ST)在图像中的面积因素，同时需要考虑目标的形状因素。一般而言，分割出的目标 ST 面积与分割参照结果 RT 面积相近时，形状因素对于分割的性能评价起主导作用，而在 ST 与 RT 有相似的形状时，面积对于分割的性能具有重要的意义。当 ST 的面积和形状与 RT 相差

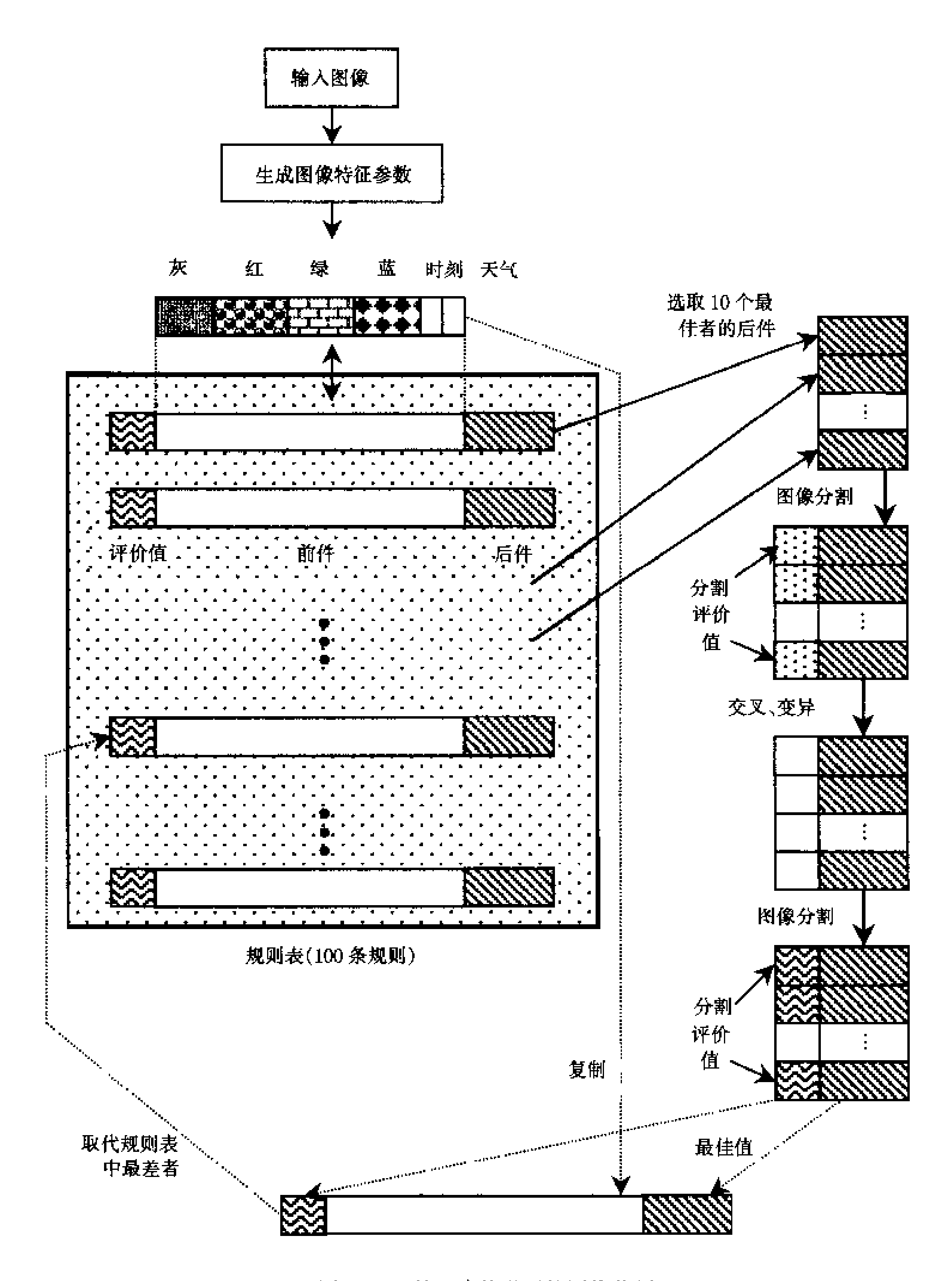


图 11.3 基于遗传学习的图像分割

均比较大时,性能评价指标具有较小的值。因此,确定分割评价指标的几个原则为:

① 如果 ST 与 RT 完全相同,则评价值为 1;

② 如果 ST 与 RT 完全不同,则评价值为 0;

③ 如果 ST 与 RT 部分相同,则评价值为在 0 至 1 之间;

④ 分割评价值由 ST 与 RT 的面积因素和形状因素共同决定。

假设面积因素、形状因素分别为 m_A 和 m_F，则分割评价指标 m 可按下式计算：

$$m = m_A m_F^{m_A} \tag{11.3}$$

m_A 为图像的背景面积 S、RT 的面积 T、分割正确的目标面积 T_c 和分割不正确的目标面积 T_i 的函数，可按下式计算：

$$m_A = (\frac{T_c}{T})^{\frac{S}{T}}(\frac{S - T_i}{S}) \tag{11.4}$$

m_F 为两个二维区域 ST 和 RT 之间形状差的函数，可按下式计算：

$$m_F = 1 - (2\sqrt{\pi}\left|\frac{\sqrt{T}}{l_T} - \frac{\sqrt{T_a}}{l_a}\right|)^{\frac{T}{S}} \tag{11.5}$$

上式中 l_T 为 RT 的周长，T_a 和 l_a 分别为 ST 的面积和周长。

11.2.2 基于遗传算法的阈值优化

1. 二维熵图像阈值分割的遗传算法

在理想情况下，图像的灰度直方图是双峰的，在这种情况下，将目标从背景中分割出来的最佳阈值就是直方图中间的谷。但在其他情况下，图像直方图会出现多峰或单峰，这时阈值的选取就出现了困难。二维熵阈值图像分割法吸取了 Shannon 的熵的概念发展而来的阈值选取方法。其原理为：设图像的灰度分为 L 级，在图像的各像素点处，像素与其右/下的像素形成一灰度二元组。设二元组 (i,j) 出现的频数为 f_{ij}，其中 i,j 分别对应此像素点的灰度值和其右/上像素灰度值。其联合概率密度定义为下式：

$$P_{ij} = \frac{f_{ij}}{M \times N} \tag{11.6}$$

式中 $M \times N$ 为图像大小。这就形成了代表灰度空间变化关系的灰度共生矩阵，如图 11.4 所示。图中 A 和 C 分别表示目标和背景，B 和 D 分别表示目标和背景的变化，用 $P_A(t)$，$P_B(t)$，$P_C(t)$ 和 $P_D(t)$ 分别表示 A,B,C,D 四个区域的取值概率。N. R. Pal 和 S. K. Pal 利用这四个概率值，定义了二次局部熵和连接熵，并提出了两种图像分割的方法，即分别求出使灰度共生矩阵的局部熵和连接熵为最大时的分割灰度 t，t 值即为图像分割的最佳阈值。

这里引入相对熵的概念用于图像分割。设 p_{ij} 和 p'_{ij} 分别为原图像和经阈值 t 分割后二值图像共生矩阵变化的概率分布，则用于衡量原图像和二值图像之间偏差的相对熵定义为：

$$L(p,p') = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p_{ij}\lg\frac{p_{ij}}{p'_{ij}} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p_{ij}\lg p_{ij} - \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p_{ij}\lg p_{ij} \tag{11.7}$$

求使 $L(p,p')$ 最小时的 t，作为图像分割的阈值。上式可简化为求 $\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p_{ij}\lg p_{ij}$ 最大。

$$\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p_{ij}\lg p_{ij} = \sum_{A} p_{ij}\lg p_{ij}^{'(A)}(t) + \sum_{B} p_{ij}\lg p_{ij}^{'(B)}(t) + \sum_{C} p_{ij}\lg p_{ij}^{'(C)}(t) + \sum_{D} p_{ij}\lg p_{ij}^{'(D)}(t) \tag{11.8}$$

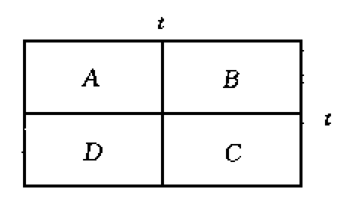


图 11.4 灰度共生矩阵平面图

由于二值图像的灰度级只有两个，分割时对于灰度大于 t 的像素赋值 l，而对于灰度小于 t 的像素赋值 0。因此，二值图

像的灰度变化按等概率处理，定义如下：

$$p_{ij}^{'(A)}(t) = q_A(t) = \frac{P_A(t)}{(t+1)^2}, \qquad (0 \leqslant i \leqslant t, 0 \leqslant j \leqslant t) \tag{11.9}$$

$$p_{ij}^{'(B)}(t) = q_B(t) = \frac{P_B(t)}{(t+1)(L-t-1)}, \qquad (0 \leqslant i \leqslant t, t+1 \leqslant j \leqslant L-1) \tag{11.10}$$

$$p_{ij}^{'(C)}(t) = q_C(t) = \frac{P_C(t)}{(L-t-1)^2}, \qquad (t+1 \leqslant i \leqslant L-1, t+1 \leqslant j \leqslant L-1) \tag{11.11}$$

$$p_{ij}^{'(D)}(t) = q_D(t) = \frac{P_D(t)}{(t+1)(L-t-1)}, \qquad (t+1 \leqslant i \leqslant L-1, 0 \leqslant j \leqslant t) \tag{11.12}$$

将以上四个象限内灰度变化概率代入(11.8)式，可得到：

$$\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} p_{ij}\lg p_{ij} = P_A \lg q_A(t) + P_B \lg q_B(t) + P_C \lg q_C(t) + P_D \lg q_D(t) \tag{11.13}$$

将二值分割推广到多阈值分割时，考虑到多阈值分割计算量较大，使用如表 11.1 所示的对称灰度共生矩阵。为求取最佳阈值($t_1, t_2, \cdots, t_n$)，只需求得下式最大即可。

表 11.1 对称灰度共生矩阵

A_{11}	A_{12}	A_{13}	$\cdots$	A_{1n}
	A_{21}	A_{22}	$\cdots$	$A_{2,n-1}$
		A_{31}	$\cdots$	$A_{3,n-2}$
			$\ddots$	$\vdots$
				A_n

$$S = \sum_{i=1}^{n} P_{A_{1i}}(t)\lg q_{A_{1i}}(t) + \sum_{i=1}^{n} P_{B_{1i}}(t)\lg q_{B_{1i}}(t) + \sum_{i=1}^{n} P_{C_{1i}}(t)\lg q_{C_{1i}}(t) + \sum_{i=1}^{n} P_{D_{1i}}(t)\lg q_{D_{1i}}(t) \tag{11.14}$$

假设将一幅图像分割成 $n+1$ 类，则有 n 个阈值。遗传算法应用于多分割阈值的搜索优化时，将待求的 n 个阈值按顺序排列起来，按二进制编码生成个体的染色体编码。应用(11.14)式对个体进行评价计算。

上述方法的实算表明，遗传算法比穷尽搜索法实现阈值寻优快，最优解对应图像分割性能好且稳定。

2．遗传优化阈值的类间方差法

最大类间方差法是一种能自动确定阈值的图像分割方法。其基本思想是：把图像中的像素按灰度值用阈值 t 分成两类 C_0 和 C_1，C_0 由灰度值 $0\sim t$ 之间的像素组成，C_1 由灰度值在 $t+1\sim L-1$ 之间的像素组成，按下式计算两类之间的类间方差：

$$\sigma(t)^2 = w_1(t)w_2(t)(u_1(t) - u_2(t))^2 \tag{11.15}$$

式中 $w_1(t)$为 C_0 中包含的像素数，$w_2(t)$为 C_1 中包含的像素数，$u_1(t)$为 C_0 中所有像素的

平均灰度值，$u_2(t)$为 C_1 中所有像素的平均灰度值。

从 0 到 $L-1$ 依次改变 t 值，取 σ 最大的 t 值为最佳阈值。

最大类间方差法的核心是计算类间方差，图像中的灰度级别越多，计算方差的次数越多，其阈值选取的时间也越长。在实际的图像处理中，往往采用局部阈值选取的方法，即把一幅图像分成若干个子块，对每一个子块单独运用最大类间方差法求取阈值。以有 256 个灰度级的大小为 1 000×1 000 的图像为例，若子块大小为 100×100，共有 100 个子块，则需要进行 25 600次方差计算。这种大量的方差计算，严重影响到图像分割任务的执行效率。

最大类间方差阈值优化过程很容易应用遗传算法来实现，并且可以提高寻优的效率。应用遗传算法时，将阈值变量编码为二进制码串，个体的适应度评价直接应用最大类间方差法的阈值判别准则函数。通过遗传算法优化获取最佳阈值为 t^*，然后再设定一个波动阈值 A，在 $[t^*-A, t^*+A]$范围内进行一次最大类间方差的求算，以获取最终的图像分割阈值。这样可以实现全局搜索与局部搜索方法的结合。一般波动阈值取图像灰度级的 10%。

11.2.3　基于遗传算法的分裂合并图像分割方法

遗传算法应用于分裂合并图像分割方法涉及到染色体编码、遗传操作设计和个体的适应度评价等问题。

1. 染色体编码

假设最大图像分割区域数目为 n，则一个个体的染色体编码为一个整数序列，即 $I_k=\{r_i \mid i=1,2,\cdots,n\}$，其中 r_i 对应于一个固定位置的分区序号，k 为个体编号。在图 11.5 中，(a)为

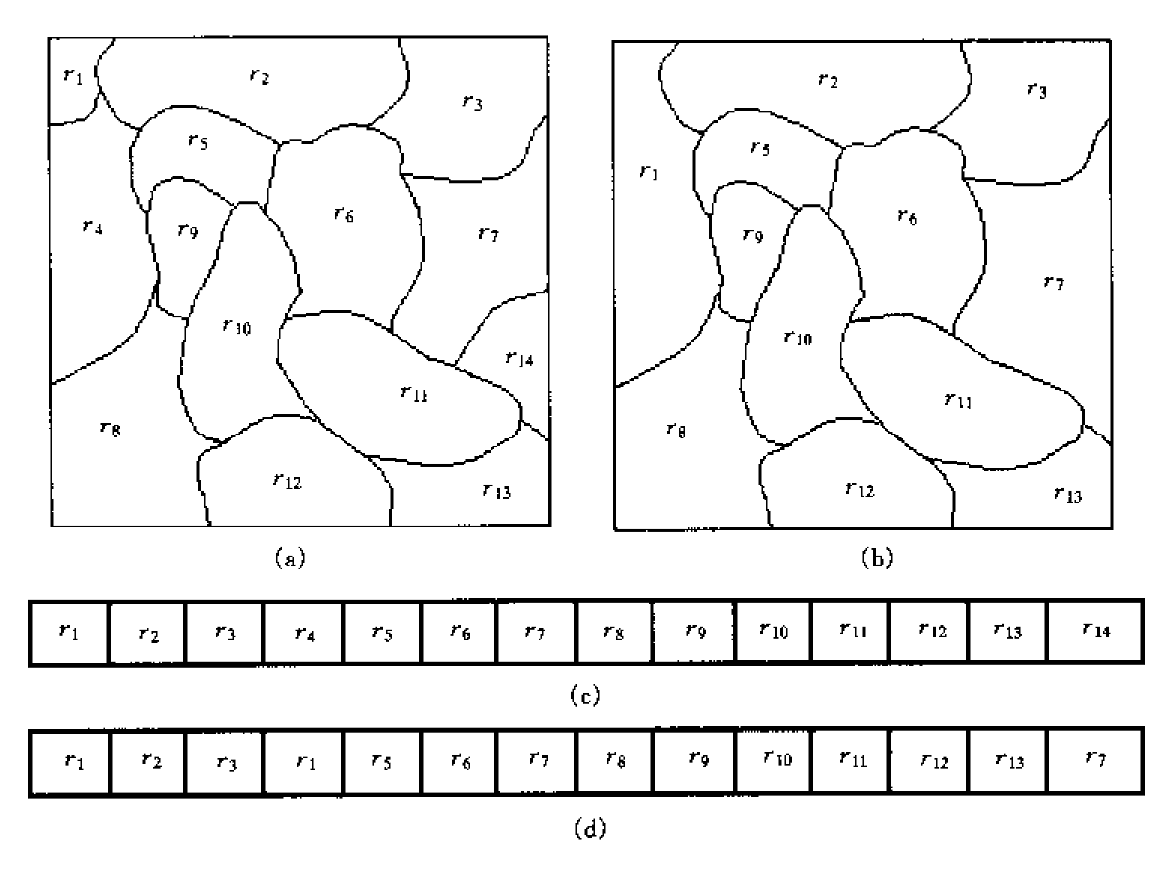


图 11.5　区域分割对应的个体染色体编码

(a) 为一个个体对应的图像分割；(b) 为(a)图中区域 r_1 与 r_4、r_7 与 r_{14}合并后的图像；(c) 为(a)图对应的染色体编码；(d) 为(b)图对应的染色体编码

一个个体的图像分割示例，则(c)为其对应的染色体编码序列。(b)为(a)图中区域 r_1 与 r_4，r_7 与 r_{14} 合并后的情况，合并后形成的个体编码发展为(d)中的编码序列。

2. 遗传操作的设计

该问题的选择操作可沿用通常的轮盘赌方法，但交叉和变异操作相对而言比较复杂一些。因为一个个体的编码序列中基因所对应的区域位置固定，交叉操作可以通过改进巡回旅行商问题中应用的 PMX 交叉法(参见 7.1 节)来实现。与旅行商问题不同的是，这里编码序列中的基因码在交叉操作后允许重复，正是依靠产生重复基因码，图像区域才得以进一步分裂或合并。此外，如果区域 r_i 与其毗邻的区域 r_j 合并，即 $I_k[i]=i$，$I_k[j]=i$，则其他已经合并到区域 r_j 的区域 r_l 也相应地合并到区域 r_i 即 $I_k[l]=i$，因此所有合并到区域 r_i 的区域基因码都是 i。下面给出两点交叉算法的伪代码：

```
Procedure 两点交叉
begin
    按轮盘赌选择两个个体 P1 和 P2，且适应度 F(P1)<F(P2)；
    if(两个个体 P1 和 P2 按概率 Pc 交叉){
    随机产生两个交叉点位置 c1 和 c2(c1<c2)；
    for(j=c1,…,c2){
       if(P2[j]<c1 or P2[j]>c2){
          for(i=1,…,c1-1,c2+1,…,n)
             if(P2[i]=P2[j])P2[i]=i;
             P2[j]=j;
       }
    else Child[j]=P2[j];
    }
    for(j=1,…,c1-1,c2+1,…,n)
       if(c1≤P1[j]≤c2)Child[j]=P2[P1[j]];
       else Child[j]=P1[j];
    }
else {
    for(j-1,…,n)
       Child[j]=P2[j];
}
end
```

图 11.6 给出了交叉操作的一个示例。子个体 *Child* 的第 5～10 位基因继承了个体 P_2，其余位的基因继承了个体 P_1。而第 11 位被改变为 r_5，第 7 位被改变为 r_7，这是上述合并处理的结果。如图 11.7 所示，(a)～(c)分别对应个体 P_1，P_2，子个体 *Child* 的区域分割。

变异操作结合局部对比度设计一种动态变异算子。所谓局部对比度是用来刻画区域边界信息模糊性的一种度量。如图 11.8 所示，区域 r_i 与邻接区域 r_j 之间距离用下式计算：

$$d_{ij}=\frac{1}{n_{ij}}\sum_{k=1}^{n_{ij}}\parallel x_{ijk}-x_{jik}\parallel^2 \tag{11.16}$$

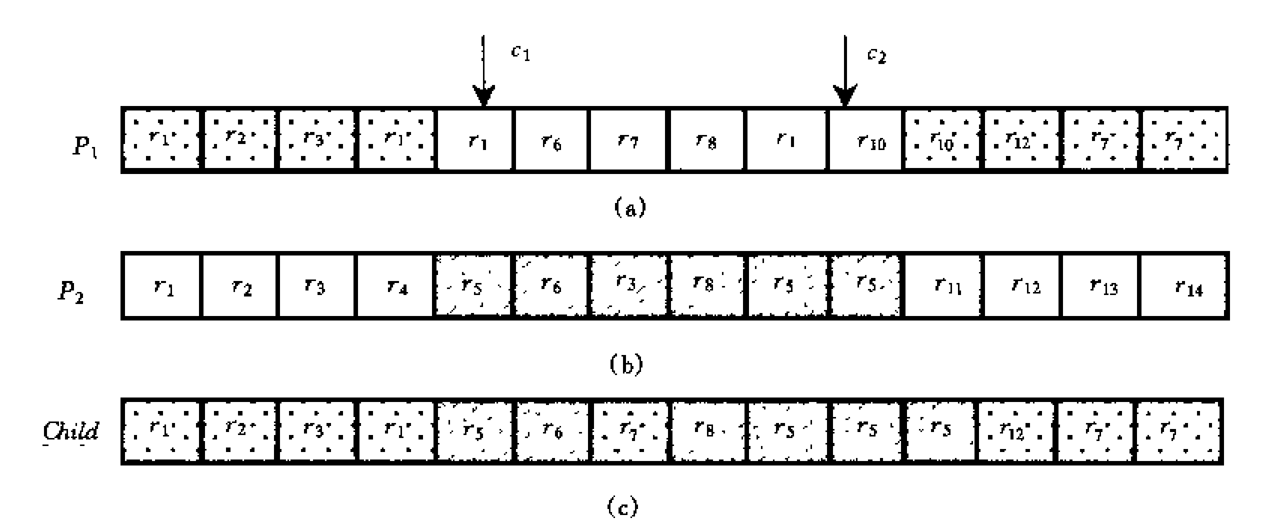


图 11.6　交叉操作

(a) 父个体 P_1；（b) 父个体 P_2；（c) 子个体 *Child*；

(a)　　(b)

(c)　　(d)

图 11.7　个体对应的区域分割示意图

(a) 个体 P_1；（b) 个体 P_2；（c) 交叉产生的子个体 *Child*；（d) *Child* 变异后生成的个体

式中 $x_{ijk}(j=1,2,\cdots,n_i;k=1,2,\cdots,n_{ij})$为区域 r_i 与邻接区域r_j 边界上第k 个像素，n_{ij}为区域r_i 与邻接区域r_j 边界上像素总数。

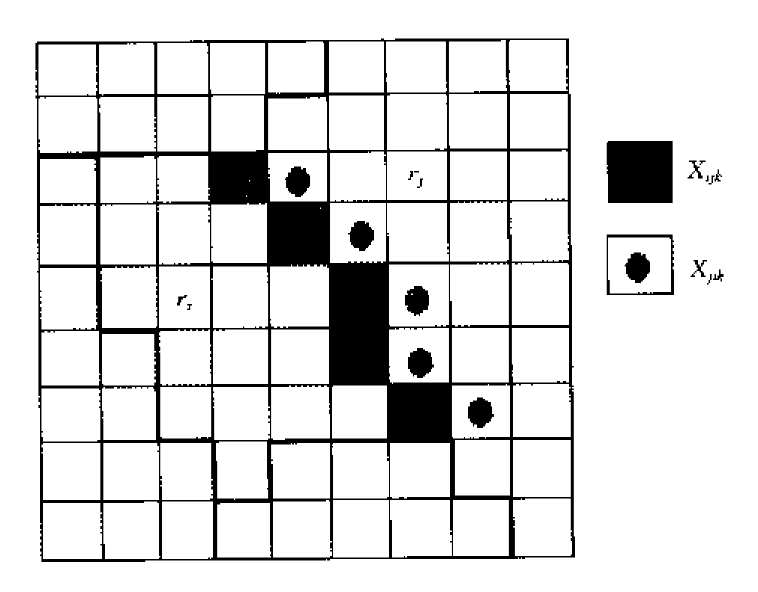


图 11.8　两个邻接区域

区域 r_i 与邻接区域r_j 之间的相对距离u_{ij}为：

$$u_{ij}=\frac{d_{ij}}{\sum_{j=1}^{n_i}d_{ij}} \tag{11.17}$$

由于 u_{ij}满足$u_{ij}\in[0,1]$，$\sum_{j=1}^{n_i}u_{ij}=1$，所以可以作为区域之间选择分裂或合并的概率。

根据上述区域之间相对距离的概念，可以定义任一区域的局部对比度 C_i 如下：

$$C_i=\begin{cases}\min\left(\dfrac{\min_j(u_{ij})}{\max_j(u_{ij})-\min_j(u_{ij})},1\right), & 若\ \min_j\neq\max_j\\ 1, & 若\ \min_j=\max_j\end{cases}$$

在一个分割对应的个体基因码中，变异操作分两个步骤。首先计算个体编码序列中所有基因码表示区域的局部对比度，按轮盘赌方法计算一个个体编码中基因变异概率，概率的大小与其对应的区域局部对比度成正比。接下来确定区域的分裂或者合并，若某区域的基因变异概率大于预定变异概率 P_m，则对该区域随机地选择发生合并或分裂。区域合并对象为与之邻接区域中相对距离 u_{ij}最小者，而分裂只发生在该区域已经与其他区域合并时，从其他区域分离出来。图 11.9 所示为一个个体变异的示例，假定第 2 位和第 5 位基因变异概率大于预定变异概率 P_m，则区域 r_5 和 r_7 被选择决定合并或分裂。若决定 r_5 合并，与 r_5 相对距离最小者为 r_2，则 r_5 并入 r_2，第 5 位基因码变为 r_2；若决定 r_7 分裂，由于此前 r_7 已与 r_{13}合并，这时将 r_7 分离出来，则第 7 位基因码不变，第 13 和第 14 位变为 r_{13}。图 11.7 中(c)和(d)分别表示了上述个体变异前后对应的图像分割。

3．个体的适应度评价

遗传算法中的一个个体对应一个图像分割，个体适应度的评价实际上是对该个体所描述的图像分割质量的衡量。在没有分割参照情况时，图像分割评价可以包括两方面的内容：区域

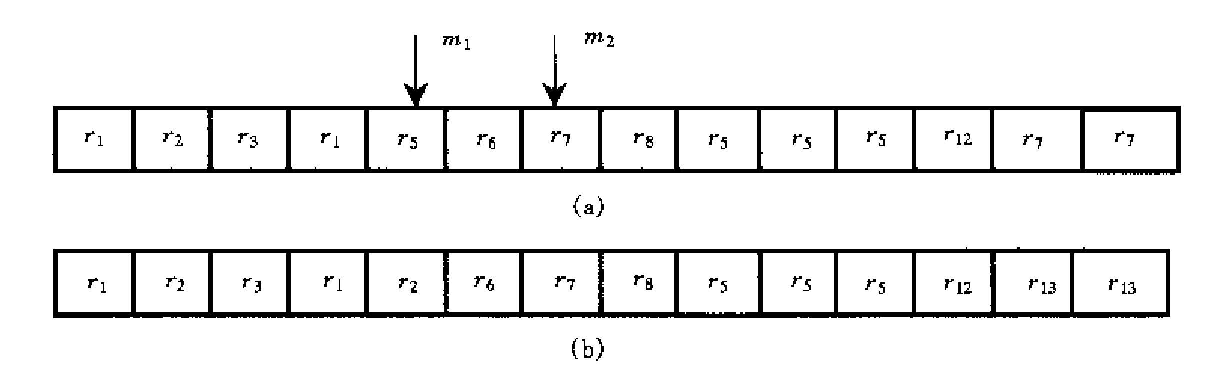


图 11.9　变异操作
(a) 父个体；(b) 变异后产生的新个体

一致性度量和边缘模糊性度量。下面我们讨论这两种度量以及遗传算法中个体适应度的计算方法。这里不考虑区域的平滑性和纹理特性。

(1) 区域一致性度量

对区域 r_i 定义如下一致性度量指标 G_i：

$$G_i = \frac{1}{n_i}\sum_{j=1}^{n_i}\frac{s_{ij}}{c_{ij}} \tag{11.18}$$

式中 s_{ij} 为区域 r_i 与邻接区域 r_j 平均灰度距离，即：

$$s_{ij} = \| m_i - m_j \|^2 \tag{11.19}$$

c_{ij} 为区域 r_i 与邻接区域 r_j 灰度方差和，即：

$$c_{ij} = \sigma_i^2 + \sigma_j^2 \tag{11.20}$$

对一个图像分割而言，一致性度量 G 为所有区域的一致性度量指标的加权和，计算公式如下：

$$G = \frac{1}{n}\sum_{i=1}^{n} w_i(p_i)G_i \tag{11.21}$$

式中，n 为区域分割数目，$w_i(p_i)$ 为区域的面积加权系数，是它区域面积 p_i 的函数，可采用下式计算：

$$w_i(p_i) = \begin{cases} \dfrac{2p_i^2}{\beta^2}, & \text{若 } \alpha \leqslant p_i < \dfrac{2}{\beta} \\ 1 - \dfrac{2(p_i - \beta)^2}{\beta^2}, & \text{若 } \dfrac{2}{\beta} \leqslant p_i < \beta \\ 1, & \text{若 } p_i \geqslant \beta \end{cases} \tag{11.22}$$

式中 α, β 分别为预定的区域最小、最大面积。

(2) 边缘模糊性度量

在前面讨论局部对比度时，我们只考虑两个邻接区域边界部分的模糊性。这里需要给出所有区域边界模糊性的综合度量 E，其计算公式如下：

$$E = \sum_{(i,j)} \frac{u_{ij}(|\nabla G|)}{u_{ij}(|\nabla G'|)} \tag{11.23}$$

式中，$|\nabla G|$ 为经分裂合并运算后图像中像素点 (i,j) 处灰度梯度，$|\nabla G'|$ 为初始区域划分中像

素点(i,j)处灰度梯度。由于$|\nabla G|\leqslant|\nabla G'|$，$E$满足$0\leqslant E\leqslant 1$。

图像中的一个像素点(i,j)处灰度梯度的计算公式如下：

$$|\nabla G(i,j)| = \left[\left(\frac{\partial G(i,j)}{\partial x}\right)_{ij}^2 + \left(\frac{\partial G(i,j)}{\partial y}\right)_{ij}^2\right]^{1/2} \tag{11.24}$$

式中，

$$\left(\frac{\partial G(i,j)}{\partial x}\right)_{ij} = \frac{1}{3}[I_{i+1,j-1} + I_{i+1,j} - (I_{i-1,j-1} + I_{i-1,j} + I_{i-1,j+1})] \tag{11.25}$$

$$\left(\frac{\partial G(i,j)}{\partial y}\right)_{ij} = \frac{1}{3}[I_{i-1,j\ 1} + I_{i,j-1} - (I_{i-1,j-1} + I_{i,j+1} + I_{i+1,j+1})] \tag{11.26}$$

遗传算法中个体的适应度评价可以采用上述两种度量的综合形式，以反映个体图像在经历一系列分裂合并过程后图像的分割质量。个体的适应度F计算为一致性度量G与边缘模糊性度量的乘积形式E，即：

$$F = GE \tag{11.27}$$

5. 图像分割实例

实例对Lena图像(128×128)用以上算法进行分割测试，种群大小为60，交叉概率P_c和变异概率P_m分别取0.2和0.1，预定分割区域最小和最大面积α，β分别取5，20。图11.10为实算时挑选出若干代中最佳个体对应的分割结果，在大约300代后达到比较好的分割质量。但在眼部、鼻部和嘴部的处理效果尚有待进一步改善。

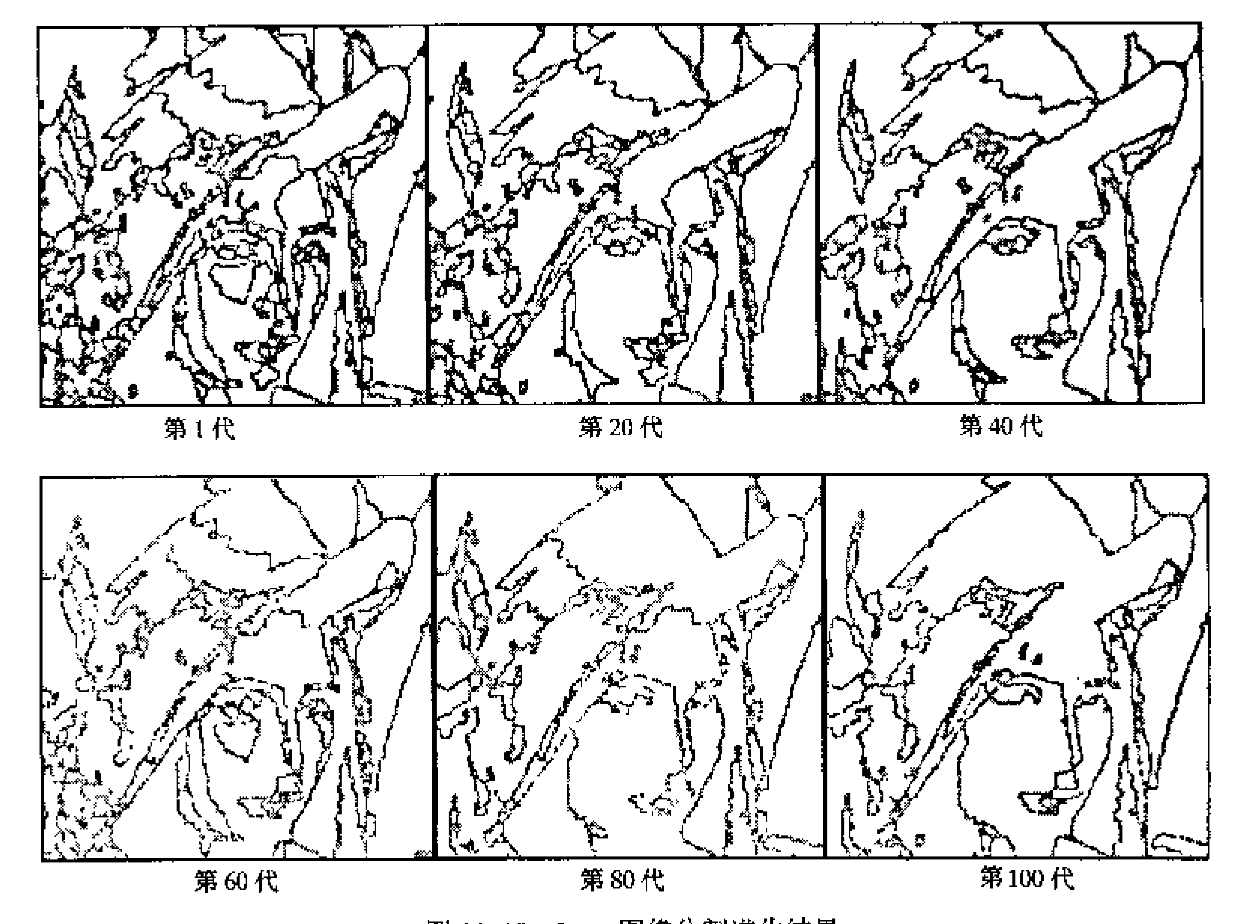
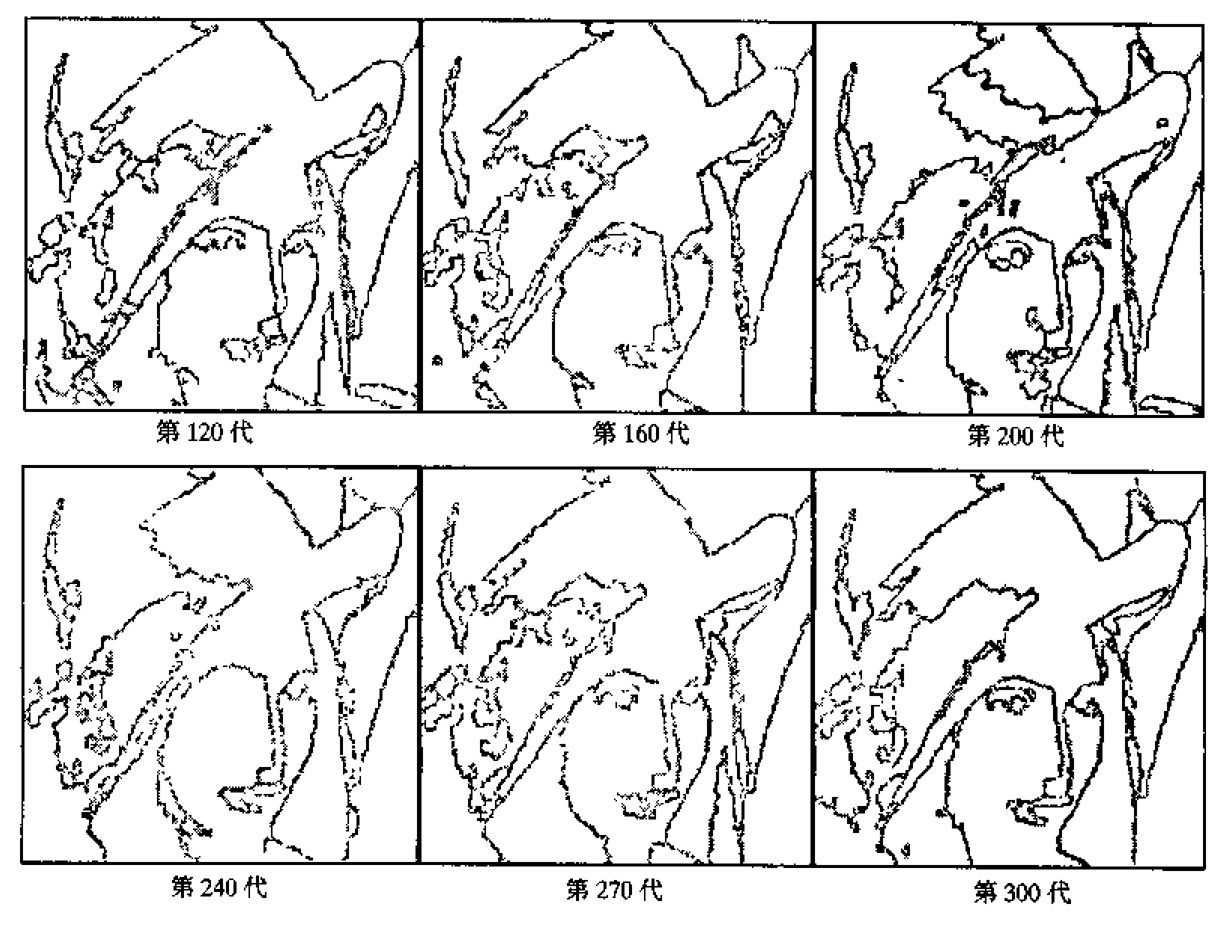


图11.10 Lena图像分割进化结果

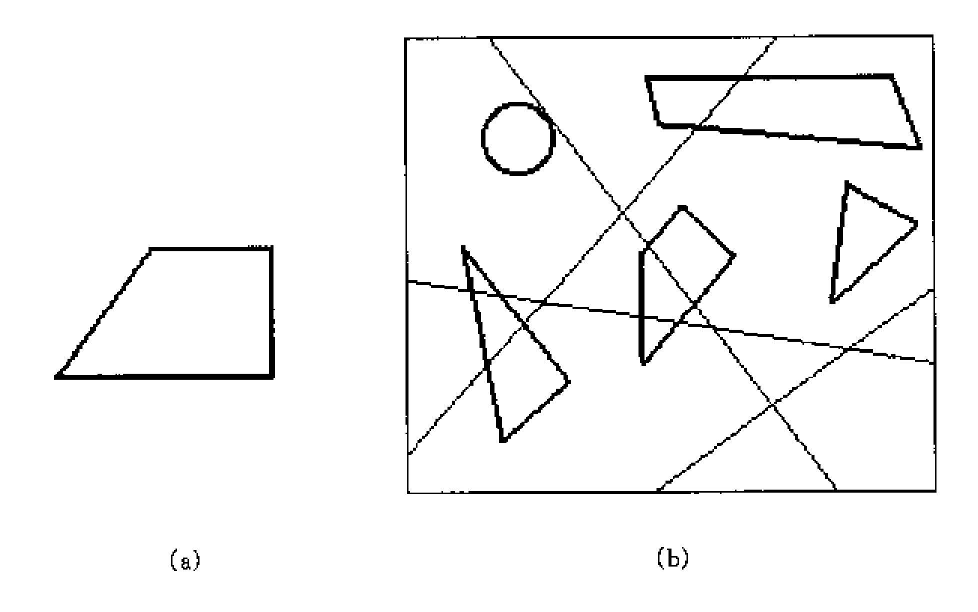


续图 11.10　Lena 图像分割进化结果

11.3　图像基元识别与提取

基元识别与提取是图像分析的重要任务。通常我们将简单的模式图形如三角形、矩形、圆、椭圆、多边形等称之为基元。如图 11.11 所示，以一个二值图像中简单模式图形的检测为例，问题是从(b)图像中检测出与模式图形(a)相似的部分。虽然类似的问题对人而言解决起来显得很简单，人们很容易从人群中发现熟悉的面孔，但目前依靠计算机解决这样的识别问题是比较困难的。因为图像中包含受噪声污染的很多不同模式，与待识别模式相似的图形自由度如位置、大小、转角等完全不能预知。这类识别问题在实际应用中具有非常重要的典型意义。

从图像中识别和提取基元一般采用著名的 Hough 变换方法(作为 1962 年美国专利成果发表)，其基本思想在于通过证据积累来提取基元。Hough 变换的主要优点是对局部缺损的不敏感性及对随机噪声的鲁棒性，因而得到了极大的关注，应用 Hough 变换可以解决三维物体识别、定位与方向检测；多传感器数据融合；雷达检测和跟踪；水声目标形状分析；多重不变图像特征抽取；识别二维图像；基于内容的图像检索；工程图中直线提取等等。由于标准的 Hough 变换的空间和时间开销很高，特别是在检测具有多参数复杂基元时效率很低。几十年来，人们作了各种努力解决此问题，如利用图像中心梯度信息减少计算量；利用递归细分参数空间和逐步增加参数空间分辨率来减少计算量和存储量；查表方法；极标编码多分辨率方法以



(a) (b)

图 11.11 二值图像中相似模式图形的检测
(a)模式图形； (b)二值图像

及并行算法等等。近年来，通过代价函数全局优化来提取基元的方法，如 Tabu 搜索法、模拟退火法、遗传算法方法得到了新的发展。早在 20 世纪 70 年代，Caricchio 就将遗传算法用于模式识别，但由于当时遗传算法尚未成熟，他仅仅是使用遗传算法设计了一组用于识别的检测器。在其后的 20 多年中遗传算法取得了巨大的进展，Roth 和 Levine 在 1992 年提出了二维和三维基元识别于提取遗传算法。

11.3.1 模式识别问题的描述

假设待识别模式可用二维点序列 P 描述为：

$$P = \{p(x_1, y_1), p(x_2, y_2), \cdots, p(x_N, y_N)\} \tag{11.28}$$

(x_i, y_i)为相对于模式原点(可视为模式图形的重心)的坐标，$p(\cdot)$为相应的灰度值，将模式放大 M 倍、旋转 θ，并将模式原点平移至(x_c, y_c)，则点序列 P 变为 Q：

$$Q = \{p(x_1^*, y_1^*), p(x_2^*, y_2^*), \cdots, p(x_N^*, y_N^*)\} \tag{11.29}$$

其中

$$\begin{pmatrix} x_j^* \\ y_j^* \end{pmatrix} = M \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}, \quad (j = 1, 2, \cdots, N) \tag{11.30}$$

对于二值图像(背景为白色，前景为黑色)而言，我们可以定义待识别模式与图像中相似图形的匹配率 R 作为识别性能的评价指标，$0 \leqslant R \leqslant 1$。

$$R = \frac{n_b}{N} \tag{11.31}$$

式中 n_b 为点序列 Q 中满足 $p(x_j^*, y_j^*) = p(x_j, y_j)$ $(j = 1, 2, \cdots, N)$的点个数。

对于灰度图像模式的匹配率 R 用两个点序列 P 和 Q 的相关系数计算。

如果匹配率 R 越大，表明识别程度越高。由此可将该识别问题转化为四维空间(x_c, y_c, M, θ)内求取匹配率 R 的最大值问题，适合用遗传算法求解。

11.3.2　基于遗传算法的基元识别与提取

(1) 染色体编码

定义遗传算法的个体 $I_k(k=1,2,\cdots)$基因型 G_k 为

$$G_k = (x_{ck}, y_{ck}, M_k, \theta_k) \tag{11.32}$$

式中$(x_{ck}, y_{ck}, M_k, \theta_k)$的定义与(11.29)式中 x_c, y_c, M, θ 含义相同,并分别以二进制进行编码。基因型 G_k 可视为四维空间中的一点,对应于表现型 H_k 为:

$$H_k = \{h_{k1}(x_{k1}^*, y_{k1}^*), h_{k2}(x_{k2}^*, y_{k2}^*), \cdots, h_{kN}(x_{kN}^*, y_{kN}^*)\} \tag{11.33}$$

式中

$$\begin{pmatrix} x_{kj}^* \\ y_{kj}^* \end{pmatrix} = M_k \cdot \begin{pmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{pmatrix} \cdot \begin{pmatrix} x_{kj} \\ y_{kj} \end{pmatrix} + \begin{pmatrix} x_{kc} \\ y_{kc} \end{pmatrix}, \quad (j=1,2,\cdots,N) \tag{11.34}$$

图 11.12 所示为计算机产生的一个包含多种基元的二值图像(含噪声),图像大小为 128×128。为简单起见,对待识别图像中相似图形的自由度作一定的限制,对$(x_{ck}, y_{ck}, M_k, \theta_k)$设定如下的取值范围:

$x_{ck} \in [0,63]$,整数

$y_{ck} \in [0,63]$,整数

$M_k \equiv 1$

$\theta_k = 45 \times n \quad (n \in [0,7]$,整数)

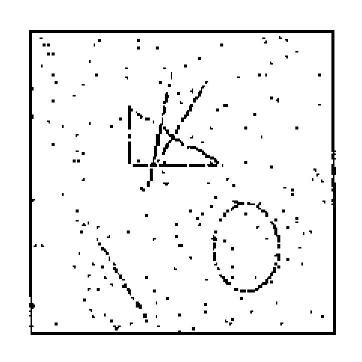


图 11.12　待识别图像示意图

在识别示例中 M_k 为 1,假定识别的相似图形大小不变,因此个体的染色体中不包含基因型 M_k。编码总长度为 15。

$$G_k = \underbrace{0\,0\,1\cdots0}_{x_{ck}(6\text{位})}\underbrace{1\,0\,1\cdots1}_{y_{ck}(6\text{位})}\underbrace{0\;\cdots\;1}_{\theta_k(3\text{位})} \tag{11.35}$$

(2) 适应度定义

个体的适应度可取匹配率指标作为评价依据。对于二值图像,由于它的噪声与信号具有相同的强度(非 0 即 1),若直接利用(11.30)式计算,将使二值图像中相匹配的个体具有极高的适应度,而在其邻域中个体的适应度很低且适应度值极为相似,从而弱化了适应度函数的导向作用,为此我们将二值识别图像进行预处理,按图 11.13 所示转化为灰阶数为 L 的多值形式。

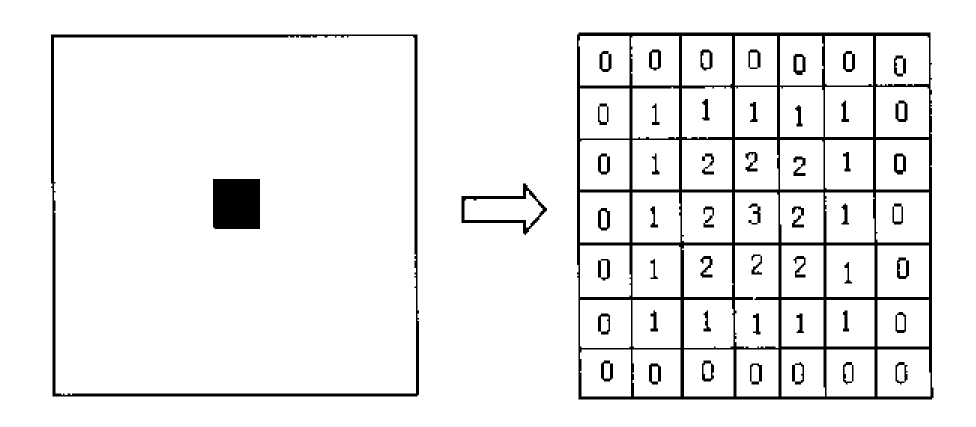


图 11.13　二值图像预处理($L=3$)

因此，个体 I_k 的适应度可按下式计算：

$$f(I_k) = \frac{\sum_{j=1}^{N} f(x_{kj}^*, y_{kj}^*)}{L \times N} \tag{11.35}$$

(3) 遗传操作的设定

遗传操作包括选择、交叉和变异，与 9.5.2 节介绍的方法相同。

(4) 实算结果分析

实现图像中对基元的识别与提取，考虑四种形状的基元：三角形、矩形、圆与椭圆、不规则形状。

实算时设定的参数有：种群大小为 10，选择操作时淘汰比例为 40%，变异率为 0.01，图像预处理灰阶数为 4，终止世代数为 120。

对图 11.12 中的图像选择检测三角形形状的图形时，如图 11.14 所示，在第 55 代后遗传算法收敛到最优解，而用随机搜索的方法很难在较短的时间内搜索到最优解。

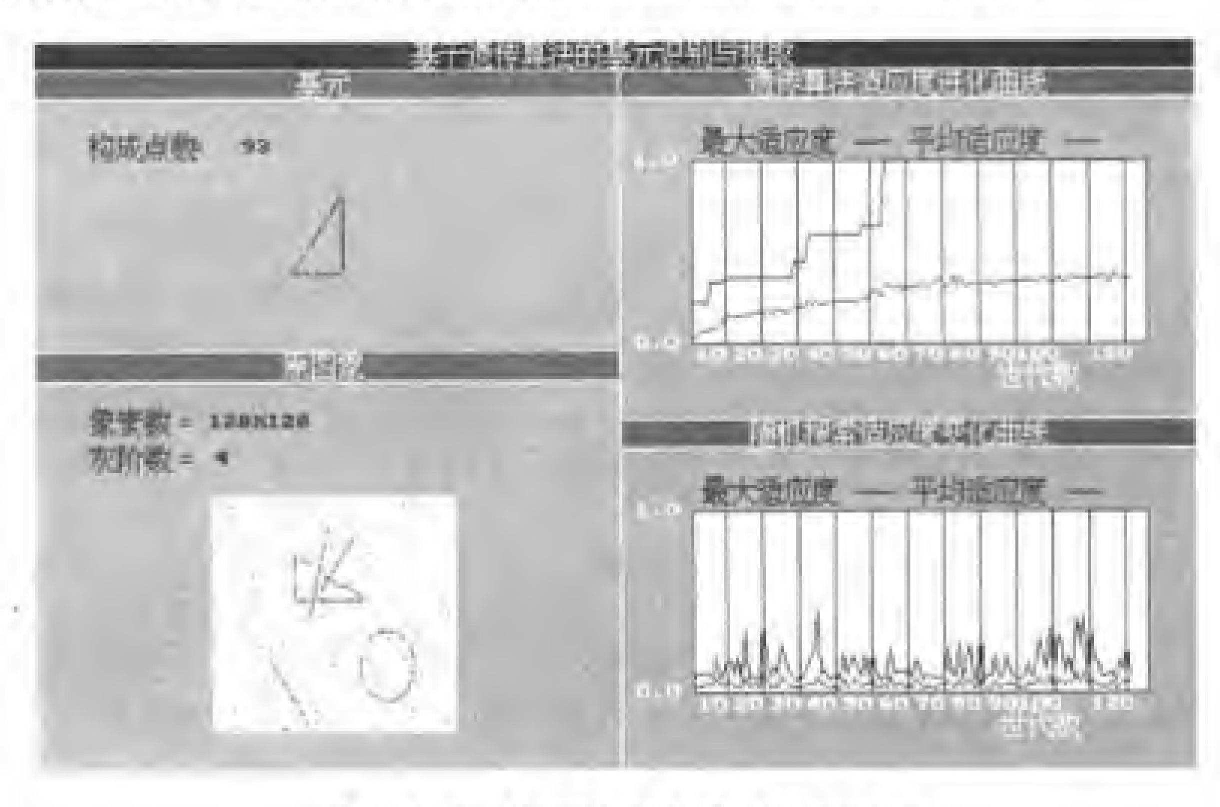


图 11.14　在二值图像中检测三角形状时实算结果

图 11.15 为在二值图像中检测不规则形状时的实算结果。

从实算结果分析可知，在图像全空间中进行基元提取的遗传算法应用于二值图像中基元识别与提取，具有尺度和旋转不变性，在噪声干扰下仍保持良好的自适应性，并可以推广到灰度图像的应用中。在同时识别和提取多个不同基元时，该问题演变为多模态函数优化问题，需要引入分享(sharing)机制，对适应度函数做适当的改进。在图像中基元图形之间存在遮掩关系时，需要结合基元边界特性分割成若干个数据切片，将这些数据切片与模式图形切片作相似度分析，从而获得适应度的评价值。

参见附录 II－3 遗传识别提取基元的源程序。

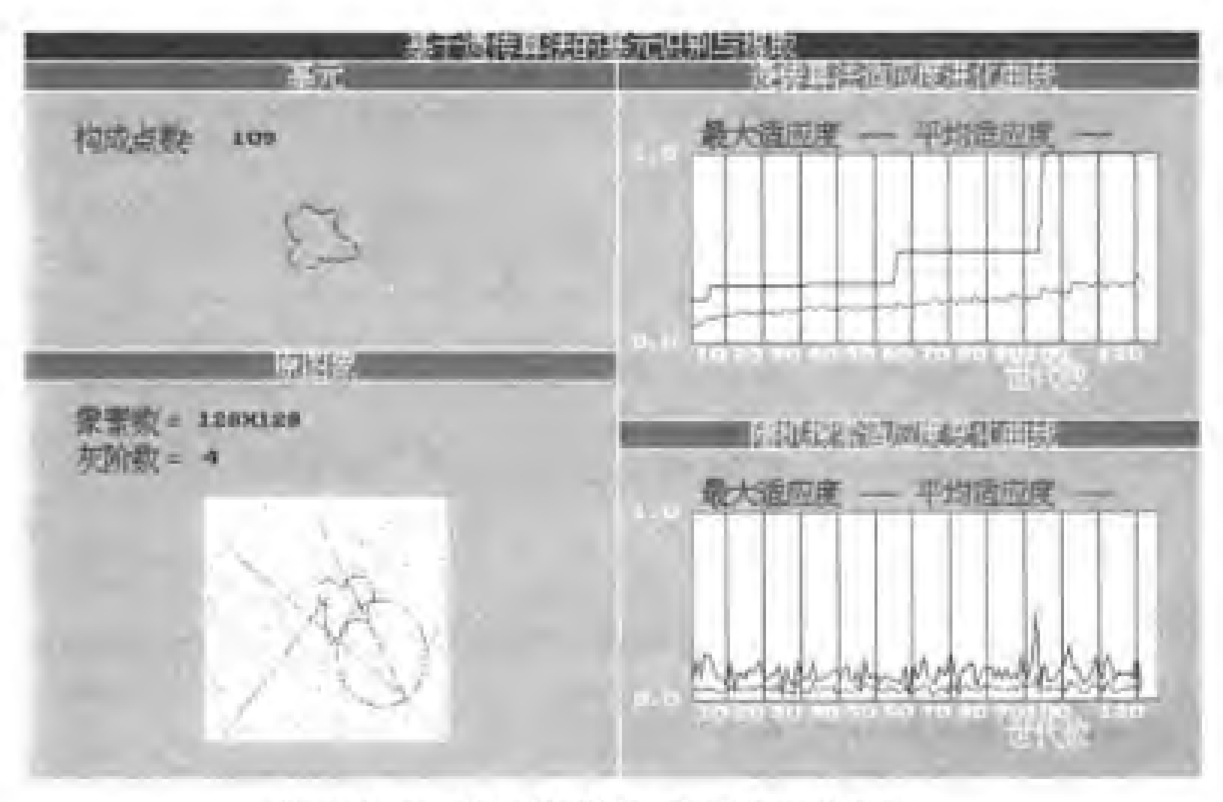


图 11.15　在二值图像中检测不规则形状时实算结果

参考文献

[1] Laws K I. The Phoenix Image Segmentation: Description and Evaluation. SRI International Technical Note, No. 289, 1982

[2] Fitzpatrick J M, Grenstettee J J, Van Gucht D. Image Restoration by Genetic Search, In: Proceedings of IEEE Southeast Conference, 1984, 460～464

[3] Bhanu B, Lee S, Ming J. Self - Optimizing Image Segmentation System Using a Genetic Algorithm. In: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1991, 362～369

[4] Takeuchi I, Furuhashi T. An Inference Method Using Multiple Patterns and Modification of Pattern Space. In: Simulated Evolution and Learning, First Asia - Pacific Conference, SEAL '96, Taejon, Korea, Springer, 1996, 187～195

[5] Chen Y W. Reconstruction of Neutron Penumbral Images by a Genetic Algorithm with Laplacian Constraint. In: Proceedings of the 1996 IEEE Signal Processing Society Workshop, Sixth in a Series of Workshops Organized by the IEEE Signal Processing Society Neural Networks Technical Committee, 1996, 100～108

[6] Chen Y W. A Parallel Genetic Algorithm Based on the Island Model for Image Restoration. In: Proceedings of the 1996 IEEE Signal Processing Society Workshop, Sixth in a Series of Workshops Organized by the IEEE Signal Processing Society Neural Networks Technical Committee, 1996, 130～139

[7] Song W K, Bien Z. Optimization of Parameters of Color Image Segmentation Using Evolutionary Programming, In: Simulated Evolution and Learning, First Asia - Pacific Conference, SEAL '96, Taejon, Korea, Springer, 1996, 116～125

[8] Nakao Z, Takashibu M. Evolutionary CT Image Reconstruction by Image Partitioning. In: Simulated Evolution and Learning, First Asia－Pacific Conference, SEAL'96, Taejon, Korea, Springer, 1996, 81～88

[9] Bhandarker S M. Image Segmentation Using Evolutionary Computation. IEEE Transactions of Evolutionary Computation, 1999, 3(1): 1～21

[10] Ser P K, Choy S T, Siu W C. Genetic Algorithm for the Extraction of Non－Analytic Objects for Multiple Dimension Parameter Space. Computer Vision and Image Understanding, 1999, 73(1): 1～13

[11] Bhandarkar S M et al. An Edge Detection Technique Using Genetic Algorithm－Based Optimization. Pattern Recognition, 1994, 27(9): 1159～1180

[12] Kuncheva L I, Fitness Functions in Editing K－NN Reference Set by Genetic Algorithms. Pattern Recognition, 1997, 30(6): 1041～1049

[13] Cross A D J et al, . Inexact Graph Matching using Genetic Search. Pattern Recognition, 1997, 30(6): 953～970

[14] Scheunders P. A Genetic C－Means Clustering Algorithm Applied to Color Image Quantification. Pattern Recognition, 1997, 30(6): 859～866

[15] Bala J et al. Shape Analysis using Hybrid Learning, Pattern Recognition, 1996, 29(8): 1323～1333

[16] Chun D N, Yang H S. Robust Image Segmentation using Genetic Algorithm with a Fuzzy Measure. Pattern Recognition, 1996, 29(7): 1 195～1 211

[17] Tackett W A. Genetic Programming for Feature Discovery and Image Discrimination. Research Report, Department of EE Systems, University of Southern California, 1995

[18] 川口剛. 遺伝的アルゴリズムによるオクルードされた物体の認識. 電子情報通信学会論文集, 1999, J82－D－II(3): 350～360

[19] 斉藤文彦. 遺伝的アルゴリズムを用いた濃淡画像の階調削減法. 電子情報通信学会論文集, 1999, J82－D－II(7): 1 140～1 149

[20] 長尾智晴. 遺伝的手法を用いたパターンマッチング. 信学技報, 1991, PRU91－92: 33～40

[21] 長尾智晴. 遺伝的アルゴリズムとその画像への応用. テレビジョン学会技術研究報告, 1992, 16(9): 1～6

[22] 章毓晋. 图像处理和分析. 北京: 清华大学出版社, 1999

[23] 边肇祺, 张学工. 模式识别(第2版). 北京: 清华大学出版社, 1999

[24] 袁占亭, 洪毅, 王青. 二维熵图像阀值分割的遗传算法. 甘肃工业大学学报, 1999, 25(1): 68～72

[25] 尹平, 王润生. 基于边缘信息的分开合并图像分割方法. 中国图像图形学报, 1998, 3(6): 450～454

[26] 郑宏, 潘励. 基于遗传算法的图像阀值的自动获取. 中国图像图形学报, 1999, 4(A)(4): 327～330

[27] 周焰, 李德华, 王祖喜 等. 三维表面三角划分的遗传算法. 中国图像图形学报, 1999, 4(A)(5): 357～361

[28] 郭国栋, 马颂德. 彩色图像分割. 中国图像图形学报, 1998, 3(11): 918～921

[29] 邓雁萍, 李介谷. 图像分割的性能评估. 模式识别与人工智能, 1996, 9(2): 144～148

[30] 刘伟权, 王明会, 钟义信. 利用遗传算法实现手写体数字识别中特征维数的压缩. 模式识别与人工智能, 1996, 9(1): 45～51

[31] 李茂军, 樊韶胜, 童调生. 单亲遗传算法在模式聚类中的应用. 模式识别与人工智能, 1999, 12(1): 32～37

[32] 罗惠韬, 章毓晋. 基于算法评价的分割算法优化思想及其系统实现. 电子科学学刊, 1998, 20(5): 577～583

[33] 王哲, 余英林. 基于马尔可夫随机场的改进遗传图像恢复方法. 华南理工大学学报, 1998, 26(1): 1～6

[34]　王熙法，杨奕若，张小俊 等．遗传算法在模式识别中的应用．小型微型计算机，1997，18(10)：32～36
[35]　张元亮，郑南宁 等．基于遗传的分形图像压缩．信息与控制，1998，27(6)：469～474
[36]　吴更石，梁德群，田原．基于遗传的分形图像压缩方法．西安交通大学学报，1999，33(4)：34～37

附录Ⅰ

有关遗传算法及进化计算的国际学术组织及其活动情况

Ⅰ.1 遗传算法、进化计算相关的学术会议

学会	会议名称	会议举办情况
国际遗传算法学会 International Society of Genetic Algorithms(ISGA)	国际遗传算法大会 International Conference on Genetic Algorithms, ICGA	1985年,1987年,1989年,1991年,1993年,1995年(美国匹兹堡),1997年(美国密歇根)
进化规划学会 Evolutionary Programming Society(EPS)	进化规划年会 Annual Conference on Evolutionary Programming(EP)	1992年,1992年,1994年,1996年(美国圣迭戈),1997年(美国印第安那)
	借助自然的并行问题求解 Parallel Problem Solving from Nature(PPSN)	1990年,1992年,1994年(比利时),1996年(德国柏林)
	遗传算法理论基础 Foundations of Genetic Algorithms(FOGA)	1991年,1993年,1994年
	进化计算会议 IEEE Conference on Evolutionary Computation	1994年,1995年,1996年(日本名古屋),1997年(美国印第安那)
	国际自适应行为模拟会议 International Conference on Simulation of Adaptive Behavior	1990年,1992年,1994年,1996年
	国际人工生命学术讨论会 International Workshop on Artificial Life	1987年,1990年,1992年(美国圣塔菲研究所),1994年(麻省理工学院),1996年(日本奈良)
	亚太模拟进化和学习会议 Asia-Pacific Conference on Simulated Evolution and Learning(SEAL)	1996年(韩国)
	国际进化算法前沿学术讨论会 International Workshop on Frontiers in Evolutionary Algorithms(FEA)	1997年(美国北卡罗里纳)
	遗传程序设计会议 Genetic Programming Conference(GP)	每年在斯坦福大学举行。

(截止到1997年年底)

I.2 知名的学术研究机构

研究机构和小组	学术带头人	主要研究内容	相关网页
美国海军人工智能应用研究中心(The Navy Center for Applied Research in Artificial Intelligence, NCARAI)		基本遗传算法和其他进化算法分析,自律小车学习策略和行为,复杂系统自适应性。	http://www.aic.nrl.navy.mil/galist
美国乔治马松大学遗传算法研究组(The Genetic Algorithms Group, George Mason University)	Dr. Kenneth De Jong	协同进化遗传算法,并行遗传算法,遗传机器学习,遗传算法分析。	
美国达特默斯大学系统分析中心(Center of Systems Analysis, University of Dortmund)	Prof. Hans-Paul Schwefel.	进化算法、突现计算和神经网络。	
美国伊利诺斯遗传算法实验室(Illinois Genetic Algorithms Laboratory, IlliGAL)	Prof. David E. Goldberg.	遗传算法理论研究与应用。	
美国苏塞克斯大学进化与自适应系统研究组(The Evolutionary and Adaptive Systems research group, the University of Sussex)		进化计算、神经网络。	
美国纳皮大学计算机系进化计算研究组(The Evolutionary Computation Research Group, the Dept. of Computer Studies, Napier University)	Terry Fogarty	进化计算及其应用,多Agent系统的进化和自调节,解决时间表问题的遗传算法等。	
美国加州大学圣地哥分校认知科学研究组(Cognitive Computer Science Research Group, University of California at San Diego)	Richard K. Belew	遗传算法	
美国密歇根州立大学遗传算法研究与应用组(The Genetic Algorithms Research and Applications Group, Michigan State University)		遗传算法、复杂自适应系统、人工生命	http://www.cps.msu.edu/wanggan1/ec.html
德国国立计算机科学研究中心自适应系统研究组(The Adaptive Systems Research Group of the German National Research Center for Computer Science (GMD)	Dr. Heinz Muehlenbein	进化算法、神经网络和机器人	http://www.systemtechnik.tu-ilmenau.de/pohlheim/GA_Toolbox/algindex.html
美国南安普顿大学机械工程系优化研究组 The Optimization Group, the Dept. of Mechanical Engineering, Southampton University		遗传算法、模拟退火、进化规划及其在结构动力学、飞机机翼设计、船舶设计中的应用	

研究机构和小组	学术带头人	主要研究内容	相关网页
美国新墨西哥大学计算机科学系 The Computer Science Department, the University of New Mexico	Prof. Stephanie Forrest.	遗传算法、计算免疫学	
英国格拉斯哥大学控制与信号系统自动设计研究组(The Group of Design Automation for Control and Signal Processing Systems, University of Glasgow)	Dr. Yun Li (李耘)	并行进化计算应用于系统辨识、自律系统、神经和模糊系统、电机驱动系统设计	http://www.mech.gla.ac.uk/yunli/
英国布鲁内大学计算机系神经和进化系统研究中心(The Centre for Neural and Evolutionary Systems (CNES), the Dept. of Computer Science, Brunel University)	Dr. Dimitri Dracopoulos	神经网络、遗传算法、遗传程序设计、人工生命、自适应系统等应用研究	http://http2.brunel.uk:8080:/research/AI/alife/ga.htm
美国圣塔菲研究所进化元胞自动机研究组 Evolving Cellular Automata (EVCA) Group, the Santa Fe Institute	Dr. Melanie Mitchell	用遗传算法进化元胞自动机应用于全局协调信息处理中的计算模型	ftp.alife.santafe.edu:/pub/USER AREA/EC
斯坦福大学(Stanford University)	Dr. John Koza	遗传程序设计	
衣荷华州立大学计算机科学系人工智能研究组(The Artificial Intelligence Research Group, the Department of Computer Science, Iowa State University)	Dr. Vasant Honavar	神经网络结构进化设计及其相关方法应用于机器人控制、学习,智能 Agent	
英国谢费尔德大学自动控制系统工程系(Evolutionary Computation Research, the Automatic Control & Systems Engineering Department, Sheffield University)	Prof. Peter Fleming.	多目标遗传算法应用于生产调度、控制系统设计等。	http://www.shef.ac.uk/uni/projects/gaipp
阿根廷布诺斯爱丽丝大学进化计算研究组(Evolutionary Computation Group, the University of Buenos Aires, Argentina)	Prof. Attilio Giordana, Prof. Lorenza Saitta	进化计算、复杂自适应系统、经济学、分布式遗传机器学习	
澳大利亚公益科学和工业科学组织信息技术部高性能计算项目组(High Performance Project, Division of Information Technology, C.S.I.R.O, Australia.)		并行遗传算法,时间表问题。	
荷兰利穆伯格大学知识系统研究所 Research Institute for Knowledge Systems, University of Limburg, the Netherlands.	Prof. Jan Paredis	协同进化遗传算法、约束满足问题、过程控制	http://lumda.matriks.rulimberg.nl/CE/paredis/

研究机构和小组	学术带头人	主要研究内容	相关网页
西班牙格兰拉达大学计算机与人工智能系(Dept. of Computer Science and Artificial Intelligence, University of Grannada, Spain.)	Prof. O. Cordon, F. Herrera	进化系统与模糊逻辑的融合	http://decsai.ugr.es/herrera/fl-ga.html
日本名古屋大学信息电子学系(Dept. of Information Electronics, Nagoya University, JAPAN)	Prof. Takshi FURRUHASHI	遗传算法、模糊控制	http://www.bioele.nuee.nagoya-u.ac.jp/member/tom/papers/icec/index.html
日本电子技术综合研究所(Eletrotechnical Laboratory, JAPAN)	Dr. Hitoshi IBA	进化计算理论	

Ⅰ.3 Internet 上的相关资源

(1) 电子邮件服务

目前,国际互联网上与进化算法相关的 EC Community 提供多种 mailing list,其中包括会员制的信息服务,服务内容包括遗传算法相关的论题讨论、会议信息及论文、研究报告、书籍出版信息。特别在欧美国家,遗传算法应用领域非常广泛,一般公开发表的有关论文、文摘均提供电子版,方便网上查阅。下面是三个会员制的电子邮件地址:

GA - List　GA - List - Requist@aic.nrl.navy.mil

EP - List　EP - List - Requist@magenta.me.fau.edu

GP - List　Genetic - Programming - Requist@cs.stanford.edu

(2) Usenet 新闻组

Usenet 上专门的遗传算法新闻组 comp.ai.genetic,该新闻组是根据威尔士大学的 Davud Beasley 所做的遗传算法 FAQ(Frequently Asked Question)经扩展后形成的,经常发布遗传算法在各学科领域的热门话题、最新会议和学术期刊提要。

(3) 在线信息服务

许多 WWW,ftp 网址提供遗传算法和进化计算论文和文摘在线服务,其中比较著名的网址如下:

GA Archive	http://www.aic.nrl.navy.mil/galist/
圣塔菲研究所进化计算宝库(ENCORE)	ftp.alife.santafe.edu:/pub/USER-AREA/EC/
国立澳大利亚大学(ANU)	ftp.life.anu.edu.au:/pub/complex_systems/alife/, http://life.anu.edu.au/
法国 LGI 研究所	ftp.imag.fr:/pub/SYMPA/
卡耐基·梅隆大学计算机系	ftp.reports.adm.cs.cmu.edu, http://www.cs.cmu.edu:8001/

Ⅰ.4 自由软件

软件名称	研制者	操作系统与环境	开发语言	主要特点	获得渠道
Genitor	Darrell Whitley	UNIX	C语言	适用于最优化问题,可选用二进制编码、实数编码,可增加新的遗传操作,可采用排序选择。	ftp.cs.colostate.edu:/pub/FENITOR.tar
Evolution Machine	Hans Michael Voigt 等	MS-DOS	Turbo-C++	适用最优化问题,采用遗传算法和进化策略。	ftp.bionik.fb10.tu-berlin.de:/pub/software
BUGS	Joshua R. Smith	UNIX, X11, Suntools	C语言	遗传算法的教学软件	ftp.santafe.edu:/pub/OLD/misc/BUGS/BUGS.tar.z
GENESIS 5.0	John J. Grefenstette	UNIX, DOS	C语言	目前最佳的遗传算法软件。另有并行遗传算法软件 dgenesis 1.0。	ftp.aic.nrl.navy.mil:/pub/galist/src/ga/genesis.tar.Z
GENEsYs-1.0	Thomas Back	UNIX	C语言	根据 GENESIS 开发的软件,增加了许多遗传操作算子的选择。	
Genetic-2N	Zbigniew Michalewicz	UNIX	C语言	适用于非线性运输问题最优化问题	Prof. Zbigniew Michalewicz Dept. of Computer Science University of North Carolina Charlotte, NC 28223, USA zbyszek@mosaic.uncc.edu
GENOCOP	Zbigniew Michalewicz	UNIX	C语言	适用于含约束条件的数值优化问题	同上
LibGA 1.00	Art Corcoran	UNIX, DOS, NeXT	C语言	遗传算法的算法库	GA-List-Request@aic.nrl.navy.mil
Pga-2.5	Peter Ross	UNIX	C语言	并行遗传算法的测试软件	GA-List-Request@aic.nrl.navy.mil
SGA-C	Rob Smith		C语言	Goldberg 的基本遗传算法的C语言版	The Clearinghouse for Genetic Algorithms The University of Alabama Department of Engineering Mechanics P.O. Box 870278 Tuscaloosa, Alabama 35487
GAOT	Christopher R. Houck	Ms-Windows, MATLAB	MATLAB	适用于数值优化问题。	chouck@eos.ncsu.edu
SGPC 1.1	Walter Alden Tackett 等	MS-DOS	C语言	遗传程序设计软件	genetic-programming-request@cs.stanford.edu

附录Ⅱ

源程序清单

Ⅱ.1 基本遗传算法源程序

```
/***********************************************************************/
/*                    基于基本遗传算法的函数最优化   SGA.C               */
/*          A Function Optimizer using Simple Genetic Algorithm        */
/*       developed from the Pascal SGA code presented by David E.Goldberg */
/*               同济大学计算机系   王小平      2000 年 5 月              */
/***********************************************************************/
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include "graph.c"
/* 全局变量 */
struct individual                     /* 个体 */
{
    unsigned   *chrom;                /* 染色体 */
    double     fitness;               /* 个体适应度 */
    double     varible;               /* 个体对应的变量值 */
    int        xsite;                 /* 交叉位置 */
    int        parent[2];             /* 父个体 */
    int        *utility;              /* 特定数据指针变量 */
};
struct bestever                       /* 最佳个体 */
{
    unsigned   *chrom;                /* 最佳个体染色体 */
    double     fitness;               /* 最佳个体适应度 */
    double     varible;               /* 最佳个体对应的变量值 */
    int        generation;            /* 最佳个体生成代 */
};
  struct individual *oldpop;           /* 当前代种群 */
  struct individual *newpop;           /* 新一代种群 */
  struct bestever bestfit;             /* 最佳个体 */
  double sumfitness;                   /* 种群中个体适应度累计 */
  double max;                          /* 种群中个体最大适应度 */
  double avg;                          /* 种群中个体平均适应度 */
  double min;                          /* 种群中个体最小适应度 */
  float  pcross;                       /* 交叉概率 */
  float  pmutation;                    /* 变异概率 */
  int     popsize;                     /* 种群大小 */
  int     lchrom;                      /* 染色体长度 */
  int     chromsize;                   /* 存储一染色体所需字节数 */
  int     gen;                         /* 当前世代数 */
  int     maxgen;                      /* 最大世代数 */
  int     run;                         /* 当前运行次数 */
  int     maxruns;                     /* 总运行次数 */
  int     printstrings;                /* 输出染色体编码的判断,0--不输出, 1--输出 */
  int     nmutation;                   /* 当前代变异发生次数 */
  int     ncross;                      /* 当前代交叉发生次数 */

/* 随机数发生器使用的静态变量 */
```

```
static double oldrand[55];
static int jrand;
static double rndx2;
static int rndcalcflag;
/* 输出文件指针 */
FILE *outfp;
/* 函数定义 */
void advance_random();
int flip(float); rnd(int, int);
void randomize();
double randomnormaldeviate();
float randomperc(), rndreal(float, float);
void warmup_random(float);
void initialize(), initdata(), initpop();
void initreport(), generation(), initmalloc();
void freeall(), nomemory(char *), report();
void writepop(), writechrom(unsigned *);
void preselect();
void statistics(struct individual *);
void title(), repchar (FILE *, char *, int);
void skip(FILE *, int);
int select();
void objfunc(struct individual *);
int crossover (unsigned *, unsigned *, unsigned *, unsigned *);
void mutation(unsigned *);


void initialize()  /* 遗传算法初始化 */
{
    /* 键盘输入遗传算法参数 */
    initdata();
    /* 确定染色体的字节长度 */
    chromsize = (lchrom/(8*sizeof(unsigned)));
    if(lchrom%(8*sizeof(unsigned))) chromsize++;
    /* 分配给全局数据结构空间 */
    initmalloc();
    /* 初始化随机数发生器 */
    randomize();
    /* 初始化全局计数变量和一些数值 */
    nmutation = 0;
    ncross = 0;
    bestfit.fitness = 0.0;
    bestfit.generation = 0;
    /* 初始化种群,并统计计算结果 */
    initpop();
    statistics(oldpop);
    initreport();
}

void initdata()  /* 遗传算法参数输入 */
{
    char answer[2];
    setcolor(9);
    setbkcolor(15);
    disp_hz16("种群大小(20-100):",100,150,20);
    gscanf(320,150,9,15,4,"%d", &popsize);
      if((popsize%2) != 0)
        {
```

```
        fprintf(outfp, "种群大小已设置为偶数\n");
        popsize++;
      };
    setcolor(9);
    setbkcolor(15);
    disp_hz16("染色体长度(8-40):", 100, 180, 20);
    gscanf(320, 180, 9, 15, 4, "%d", &lchrom);
    setcolor(9);
    setbkcolor(15);
    disp_hz16("是否输出染色体编码(y/n):", 100, 210, 20);
    printstrings=1;
    gscanf(320, 210, 9, 15, 4, "%s", answer);
      if(strncmp(answer, "n", 1) == 0) printstrings = 0;
    setcolor(9);
    setbkcolor(15);
    disp_hz16("最大世代数(100-300):", 100, 240, 20);
    gscanf(320, 240, 9, 15, 4, "%d", &maxgen);
    setcolor(9);
    setbkcolor(15);
    disp_hz16("交叉率(0.2-0.9):", 100, 270, 20);
    gscanf(320, 270, 9, 15, 5, "%f", &pcross);
    setcolor(9);
    setbkcolor(15);
    disp_hz16("变异率(0.01-0.1):", 100, 300, 20);
    gscanf(320, 300, 9, 15, 5, "%f", &pmutation);
}

void initpop()  /* 随机初始化种群 */
{
    int j, j1, k, stop;
    unsigned mask = 1;
    for(j = 0; j < popsize; j++)
    {
      for(k = 0; k < chromsize; k++)
      {
        oldpop[j].chrom[k] = 0;
        if(k == (chromsize-1))
          stop = lchrom - (k*(8*sizeof(unsigned)));
        else
          stop =8*sizeof(unsigned);
        for(j1 = 1; j1 <= stop; j1++)
        {
          oldpop[j].chrom[k] = oldpop[j].chrom[k]<<1;
          if(flip(0.5))
            oldpop[j].chrom[k] = oldpop[j].chrom[k]|mask;
        }
      }
      oldpop[j].parent[0] = 0;  /* 初始父个体信息 */
      oldpop[j].parent[1] = 0;
      oldpop[j].xsite = 0;
      objfunc(&(oldpop[j]));  /* 计算初始适应度 */
    }
}

void initreport()  /* 初始参数输出 */
{
    void    skip();
```

```
    skip(outfp, 1);
    fprintf(outfp, "              基本遗传算法参数\n");
    fprintf(outfp, " -------------------------------------------------------\n");
    fprintf(outfp, "    种群大小(popsize)        = %d\n", popsize);
    fprintf(outfp, "    染色体长度(lchrom)       = %d\n", lchrom);
    fprintf(outfp, "    最大进化代数(maxgen)     = %d\n", maxgen);
    fprintf(outfp, "    交叉概率(pcross)         = %f\n", pcross);
    fprintf(outfp, "    变异概率(pmutation)      = %f\n", pmutation);
    fprintf(outfp, " -------------------------------------------------------\n");
    skip(outfp, 1);
    fflush(outfp);
}

void generation()
{
  int mate1, mate2, jcross, j = 0;
  /* 每代运算前进行预选 */
  preselect();
  /* 选择, 交叉, 变异 */
  do
    {
      /* 挑选交叉配对 */
      mate1 = select();
      mate2 = select();
      /* 交叉和变异 */
      jcross = crossover(oldpop[mate1].chrom, oldpop[mate2].chrom, newpop[j].chrom, newpop[j+1].chrom);
      mutation(newpop[j].chrom);
      mutation(newpop[j+1].chrom);
      /* 解码, 计算适应度 */
      objfunc(&(newpop[j]));
      /* 记录亲子关系和交叉位置 */
      newpop[j].parent[0] = mate1+1;
      newpop[j].xsite = jcross;
      newpop[j].parent[1] = mate2+1;
      objfunc(&(newpop[j+1]));
      newpop[j+1].parent[0] = mate1+1;
      newpop[j+1].xsite = jcross;
      newpop[j+1].parent[1] = mate2+1;
      j = j + 2;
    }
  while(j < (popsize-1));

}

void initmalloc()   /* 为全局数据变量分配空间 */
{
  unsigned nbytes;
  char *malloc();
  int j;
  /* 分配给当前代和新一代种群内存空间 */
  nbytes = popsize*sizeof(struct individual);
  if((oldpop = (struct individual *) malloc(nbytes)) == NULL)
    nomemory("oldpop");
  if((newpop = (struct individual *) malloc(nbytes)) == NULL)
    nomemory("newpop");
  /* 分配给染色体内存空间 */
  nbytes = chromsize*sizeof(unsigned);
  for(j = 0; j < popsize; j++)
```

```
    }
      if((oldpop[j].chrom = (unsigned *) malloc(nbytes)) == NULL)
      nomemory("oldpop chromosomes");
      if((newpop[j].chrom = (unsigned *) malloc(nbytes)) == NULL)
      nomemory("newpop chromosomes");
    }
  if((bestfit.chrom = (unsigned *) malloc(nbytes)) == NULL)
      nomemory("bestfit chromosome");

}

void freeall()  /* 释放内存空间 */
{
  int i;
    for(i = 0; i < popsize; i++)
      {
        free(oldpop[i].chrom);
        free(newpop[i].chrom);
      }
  free(oldpop);
  free(newpop);
  free(bestfit.chrom);
    }

void nomemory(string)  /* 内存不足,退出 */
  char *string;
{
  fprintf(outfp, "malloc: out of memory making %s!! \n", string);
  exit(-1);
}

void report()  /* 输出种群统计结果 */
{
    void repchar(), skip();
    void writepop(), writestats();
    repchar(outfp, "-", 80);
    skip(outfp, 1);
    if(printstrings == 1)
    {
        repchar(outfp, " ", ((80-17)/2));
        fprintf(outfp, "模拟计算统计报告 \n");
        fprintf(outfp, "世代数 %3d", gen);
        repchar(outfp, " ", (80-28));
        fprintf(outfp, "世代数 %3d\n", (gen+1));
        fprintf(outfp, "个体 染色体编码");
        repchar(outfp, " ", lchrom-5);
        fprintf(outfp, "适应度     父个体   交叉位置   ");
        fprintf(outfp, "染色体编码 ");
        repchar(outfp, " ", lchrom-5);
        fprintf(outfp, "适应度\n");
        repchar(outfp, "-", 80);
        skip(outfp, 1);
        writepop(outfp);
        repchar(outfp, "-", 80);
        skip(outfp, 1);
    }
    fprintf(outfp, "第 %d 代统计: \n", gen);
    fprintf(outfp, "总交叉操作次数 = %d, 总变异操作数 = %d\n", ncross, nmutation);
```

```
    fprintf(outfp, " 最小适应度：%f 最大适应度：%f 平均适应度 %f\n", min, max, avg);
    fprintf(outfp, " 迄今发现最佳个体 -> 所在代数：%d ", bestfit.generation);
    fprintf(outfp, " 适应度：%f 染色体：", bestfit.fitness);
    writechrom((&bestfit)->chrom);
    fprintf(outfp, " 对应的变量值：%f", bestfit.varible);
    skip(outfp, 1);
    repchar(outfp, "-", 80);
    skip(outfp, 1);
}

void writepop()
{
    struct individual *pind;
    int j;
    for(j=0; j<popsize; j++)
    {
        fprintf(outfp, "%3d) ", j+1);
        /* 当前代个体 */
        pind = &(oldpop[j]);
        writechrom(pind->chrom);
        fprintf(outfp, " %8f | ", pind->fitness);
        /* 新一代个体 */
        pind = &(newpop[j]);
        fprintf(outfp, "(%2d, %2d) %2d ",
        pind->parent[0], pind->parent[1], pind->xsite);
        writechrom(pind->chrom);
        fprintf(outfp, " %8f\n", pind->fitness);
    }
}

void writechrom(chrom)  /* 输出染色体编码 */
unsigned *chrom;
{
    int j, k, stop;
    unsigned mask = 1, tmp;
    for(k = 0; k < chromsize; k++)
    {
        tmp = chrom[k];
        if(k == (chromsize-1))
            stop = lchrom - (k*(8*sizeof(unsigned)));
        else
            stop = 8*sizeof(unsigned);
        for(j = 0; j < stop; j++)
        {
            if(tmp&mask)
                fprintf(outfp, "1");
            else
                fprintf(outfp, "0");
            tmp = tmp>>1;
        }
    }
}

void preselect()
{
    int j;
    sumfitness = 0;
    for(j = 0; j < popsize; j++) sumfitness += oldpop[j].fitness;
```

```
}

int select()  /* 轮盘赌选择 */
{
    extern float randomperc();
    float sum, pick;
    int i;
    pick = randomperc();
    sum = 0;
    if(sumfitness != 0)
    {
        for(i = 0; (sum < pick) && (i < popsize); i++)
            sum += oldpop[i].fitness/sumfitness;
    }
    else
        i = rnd(1,popsize);
    return(i-1);
}

void statistics(pop)  /* 计算种群统计数据 */
struct individual *pop;
{
    int i, j;
    sumfitness = 0.0;
    min = pop[0].fitness;
    max = pop[0].fitness;
    /* 计算最大、最小和累计适应度 */
    for(j = 0; j < popsize; j++)
    {
        sumfitness = sumfitness + pop[j].fitness;
        if(pop[j].fitness > max) max = pop[j].fitness;
        if(pop[j].fitness < min) min = pop[j].fitness;
        /* new global best-fit individual */
        if(pop[j].fitness > bestfit.fitness)
        {
          for(i = 0; i < chromsize; i++)
            bestfit.chrom[i]       = pop[j].chrom[i];
              bestfit.fitness      = pop[j].fitness;
              bestfit.varible      = pop[j].varible;
              bestfit.generation   = gen;
        }
    }
    /* 计算平均适应度 */
    avg = sumfitness/popsize;
}

void title()
{
  settextstyle(0,0,4);
  gprintf(110,15,4,0,"SGA Optimizer");
  setcolor(9);
  disp_hz24("基本遗传算法",220,60,25);
}

void repchar (outfp, ch, repcount)
FILE *outfp;
char *ch;
int repcount;
```

```
{
    int j;
    for (j = 1; j <= repcount; j++) fprintf(outfp, "%s", ch);
}

void skip(outfp, skipcount)
FILE *outfp;
int skipcount;
{
    int j;
    for (j = 1; j <= skipcount; j++) fprintf(outfp, "\n");
}

void objfunc(critter)  /* 计算适应度函数值 */
struct individual *critter;
{
    unsigned mask=1;
    unsigned bitpos;
    unsigned tp;
    double pow(), bitpow ;
    int j, k, stop;
    critter->varible = 0.0;
    for(k = 0; k < chromsize; k++)
    {
        if(k == (chromsize-1))
            stop = lchrom-(k*(8*sizeof(unsigned)));
        else
            stop =8*sizeof(unsigned);
        tp = critter->chrom[k];
        for(j = 0; j < stop; j++)
        {
            bitpos = j + (8*sizeof(unsigned))*k;
            if((tp&mask) == 1)
            {
                bitpow = pow(2.0, (double) bitpos);
                critter->varible = critter->varible + bitpow;
            }
            tp = tp>>1;
        }
    }
    critter->varible = -1+critter->varible*3/(pow(2.0, (double)lchrom)-1);
    critter->fitness = critter->varible*sin(critter->varible*10*atan(1)*4)+2.0;
}

void mutation(unsigned *child)  /* 变异操作 */
{
    int j, k, stop;
    unsigned mask, temp = 1;
    for(k = 0; k < chromsize; k++)
    {
        mask = 0;
        if(k == (chromsize-1))
            stop = lchrom - (k*(8*sizeof(unsigned)));
        else
            stop = 8*sizeof(unsigned);
        for(j = 0; j < stop; j++)
        {
            if(flip(pmutation))
```

```
            {
                mask = mask|(temp<<j);
                nmutation++;
            }
        }
        child[k] = child[k]^mask;
    }
}

int crossover (unsigned *parent1, unsigned *parent2, unsigned *child1, unsigned *child2)
/* 由两个父个体交叉产生两个子个体 */
{
    int j, jcross, k;
    unsigned mask, temp;
    if(flip(pcross))
    {
        jcross = rnd(1,(lchrom - 1));    /* Cross between 1 and l-1 */
        ncross++;
        for(k = 1; k <= chromsize; k++)
        {
            if(jcross >= (k*(8*sizeof(unsigned))))
            {
                child1[k-1] = parent1[k-1];
                child2[k-1] = parent2[k-1];
            }
            else if((jcross < (k*(8*sizeof(unsigned)))) && (jcross > ((k-1)*(8*sizeof(unsigned)))))
            {
                mask = 1;
                for(j = 1; j <= (jcross-1-((k-1)*(8*sizeof(unsigned)))); j++)
                {
                    temp = 1;
                    mask = mask<<1;
                    mask = mask|temp;
                }
                child1[k-1] = (parent1[k-1]&mask)|(parent2[k-1]&(~mask));
                child2[k-1] = (parent1[k-1]&(~mask))|(parent2[k-1]&mask);
            }
                else
            {
                child1[k-1] = parent2[k-1];
                child2[k-1] = parent1[k-1];
            }
        }
    }
    else
    {
        for(k = 0; k < chromsize; k++)
        {
            child1[k] = parent1[k];
            child2[k] = parent2[k];
        }
        jcross = 0;
    }
    return(jcross);
}

void advance_random()  /* 产生55个随机数 */
{
```

```
    int j1;
    double new_random;
    for(j1 = 0; j1 < 24; j1 + + )
    {
        new_random = oldrand[j1] - oldrand[j1+31];
        if(new_random < 0.0) new_random = new_random + 1.0;
        oldrand[j1] = new_random;
    }
    for(j1 = 24; j1 < 55; j1 + + )
    {
        new_random = oldrand [j1] - oldrand [j1-24];
        if(new_random < 0.0) new_random = new_random + 1.0;
        oldrand[j1] = new_random;
    }
}

int flip(float prob)  /* 以一定概率产生 0 或 1 */
{
    float randomperc();
    if(randomperc() < = prob)
        return(1);
    else
        return(0);
}

void randomize()  /* 设定随机数种子并初始化随机数发生器 */
{
    float randomseed;
    int j1;
    for(j1=0; j1<=54; j1 + + )
      oldrand[j1] = 0.0;
    jrand=0;
      do
        {
          setcolor(9);
          setbkcolor(15);
          disp_hz16("随机数种子[0-1]:", 100, 330, 20);
          gscanf(320, 330, 9, 15, 4, "%f", &randomseed);
        }
        while((randomseed < 0.0) || (randomseed > 1.0));
    warmup_random(randomseed);
}

double randomnormaldeviate()  /* 产生随机标准差 */
{
    double sqrt(), log(), sin(), cos();
    float randomperc();
    double t, rndx1;
    if(rndcalcflag)
    {  rndx1 = sqrt( - 2.0 * log((double) randomperc()));
       t = 6.2831853072 * (double) randomperc();
       rndx2 = rndx1 * sin(t);
       rndcalcflag = 0;
       return(rndx1 * cos(t));
    }
    else
    {
       rndcalcflag = 1;
```

```
        return(rndx2);
    }
}

float randomperc()  /* 与库函数random()作用相同，产生[0,1]之间一个随机数 */
{
    jrand++;
    if(jrand >= 55)
    {
        jrand = 1;
        advance_random();
    }
    return((float) oldrand[jrand]);
}

int rnd(low, high)  /* 在整数low和high之间产生一个随机整数 */
int low, high;
{
    int i;
    float randomperc();
    if(low >= high)
        i = low;
    else
    {
        i = (randomperc() * (high - low + 1)) + low;
        if(i > high) i = high;
    }
    return(i);
}


void warmup_random(float random_seed)  /* 初始化随机数发生器 */
{
    int j1, ii;
    double new_random, prev_random;

    oldrand[54] = random_seed;
    new_random = 0.000000001;
    prev_random = random_seed;
    for(j1 = 1 ; j1 <= 54; j1++)
    {
        ii = (21 * j1) % 54;
        oldrand[ii] = new_random;
        new_random = prev_random - new_random;
        if(new_random<0.0) new_random = new_random + 1.0;
        prev_random = oldrand[ii];
    }
    advance_random();
    advance_random();
    advance_random();
    jrand = 0;
}


main(argc, argv)  /* 主程序 */
int argc;
char *argv[];
```

```
}
    struct individual  * temp;
    FILE    * fopen();
    void   title();
    char    * malloc();
        if((outfp = fopen(argv[1],"w")) == NULL)
        {
          fprintf(stderr,"Cannot open output file %s\n",argv[1]);
          exit(-1);
        }
    g_init();
    setcolor(9);
    setbkcolor(15);
    title();
    disp_hz16("输入遗传算法执行次数(1-5):",100,120,20);
    gscanf(320,120,9,15,4,"%d",&maxruns);
    for(run=1; run<=maxruns; run++)
    {
        initialize();
        for(gen=0; gen<maxgen; gen++)
        {
          fprintf(outfp,"\n第 %d / %d 次运行: 当前代为 %d, 共 %d 代\n", run,maxruns,gen,maxgen);
            /* 产生新一代 */
            generation();
            /* 计算新一代种群的适应度统计数据 */
            statistics(newpop);
            /* 输出新一代统计数据 */
            report();
            temp = oldpop;
            oldpop = newpop;
            newpop = temp;
        }
        freeall();
    }
}
```

Ⅱ.2 基本遗传学习分类系统源程序

```
/*********************************************************************/
/*                  基本遗传学习分类系统 SCS.CPP                       */
/*          A Simple Classifier System based on Genetic Learning     */
/*   developed from the Pascal SCS code presented by David E. Goldberg */
/*            同济大学计算机系  王小平       2000年7月                  */
/*********************************************************************/

#include <stdlib.h>
#include <fstream.h>
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
#include <string.h>

const int wildcard = -1;              /* 通配符# */
const long iterationsperblock = 10000;
const int fmaxmating = 10;
int rndcalcflag;
double rndx2;

struct classtype                      /* 分类器 */
{
  int *c;                             /* condition */
  int a;                              /* action */
  float strength, bid, ebid;          /* 权值,投标,有效投标 */
  int matchflag;                      /* 匹配标志 */
  int specificity;                    /* 非#码长度 */
};
struct classlist                      /* 匹配表 */
{
  int *clist;                         /* 当前与环境消息匹配的分类器表 */
  int nactive;                        /* 当前匹配数目 */
};
struct poptype                        /* 分类器表数据 */
{
    struct classtype *classifier;     /* 分类器表 */
    int nclassifier, nposition;       /* 分类器数目,条件码长 */
    float pgeneral, cbid, bidsigma, bidtax, lifetax, bid1, bid2, ebid1, ebid2;
    float sumstrength, fmaxstrength, avgstrength, fminstrength;
};
struct trecord                        /* 时间参数 */
{
    int initialiteration, initialblock, iteration, block, reportperiod;
    int gaperiod, consolereportperiod, plotreportperiod, nextplotreport;
    int nextconsolereport, nextreport, nextga;
    int reportflag, gaflag, consolereportflag, plotreportflag;
};
struct erecord                        /* 环境参数 */
{
    int laddress, ldata, lsignal, address, output, classifieroutput;
    int *signal;
};
struct crecord                        /* 交易所参数 */
{
```

```
    int winner, oldwinner;
    int bucketbrigadeflag;
};
struct rrecord                    /* 增强参数(得分兑现) */
{
float reward, rewardcount, totalcount, count50, rewardcount50;
float proportionreward, proportionreward50;
int lastwinner;
};
struct mrecord                    /* 交配参数 */
{
  int mate1, mate2, mort1, mort2, sitecross;
};
struct grecord                    /* 遗传算法参数 */
{
float populationselect, pmutation, pcrossover;
long ncrossover, nmutation;
int crowdingfactor, crowdingsubpop, nselect;
struct mrecord *mating;
};

/* 全局变量 */
int *envmessage;                  /* 消息 */
struct poptype population;        /* 分类器表数据 */
struct classlist matchlist;       /* 匹配表 */
struct trecord timekeeprec;       /* 时间参数 */
struct erecord environrec;        /* 环境参数 */
struct crecord clearingrec;       /* 交易所参数 */
struct rrecord reinforcementrec;  /* 增强参数 */
struct grecord garec;             /* 遗传算法参数 */

struct poptype *pop=&population;
struct classlist *matchl=&matchlist;

ofstream rep("c:\\scs\\rep.txt");                          /* 结果报告文件 */
ofstream pfile("c:\\scs\\pfile.txt");                      /* 绘图数据文件 */
ifstream tfile("c:\\scs\\tfile.txt",ios::nocreate);        /* 时间参数文件 */
ifstream efile("c:\\scs\\efile.txt",ios::nocreate);        /* 环境参数文件 */
ifstream cfile("c:\\scs\\cfile.txt",ios::nocreate);        /* 分类器参数文件 */
ifstream rfile("c:\\scs\\rfile.txt",ios::nocreate);        /* 增强参数文件 */
ifstream gfile("c:\\scs\\gfile.txt",ios::nocreate);        /* 遗传算法参数文件 */

void initrepheader();
void interactiveheader();
void initialization();
void detectors(struct erecord *environrec, int *envmessage);
void reportdetectors(int *envmessage, int nposition);
void report();
void consolereport(struct rrecord *reinforcementrec);
void plotreport(struct rrecord *reinforcementrec);
int  addtime(int t, int dt, int &carryflag);
void inittimekeeper(struct trecord *timekeeprec);
void initreptimekeeper(struct trecord *timekeeprec);
void timekeeper(struct trecord *timekeeprec);
void reporttime(struct trecord *timekeeprec);
void generatesignal(struct erecord *environrec);
int  decode(int *mess, int start, int length);
void multiplexeroutput(struct erecord *environrec);
```

```
void environment(struct erecord * environrec);
void initenvironment(struct erecord * environrec);
void initrepenvironment(struct erecord * environrec);
void writesignal(int * signal, int lsignal);
void reportenvironment(struct erecord * environrec);
int   randomchar(double pgeneral);
void readcondition(int * c, float pgeneral, int nposition);
void readclassifier(struct classtype * Class, float pgeneral, int nposition);
int   counterspecificity(int * c,  int nposition);
void initclassifiers(struct poptype * population);
void initrepclassifiers(struct poptype * population);
void writecondition(int c[], int nposition);
void writeclassifier(struct classtype * Class, int number, int nposition);
void reportclassifiers(struct poptype * population);
int   match(int * c, int * m,  int nposition);
void matchclassifiers(struct poptype * population, int * emess, struct classlist * matchlist);
void initaoc(struct crecord * clearingrec);
void initrepaoc(struct crecord * clearingrec);
int   auction(struct poptype * population, struct classlist * matchlist, int oldwinner);
void clearinghouse(struct poptype * population, struct crecord * clearingrec);
void taxcollector(struct poptype * population);
void reportaoc(struct crecord * clearingrec);
void aoc(struct poptype * population, struct classlist * matchlist, struct crecord * clearingrec);
void initreinforcement(struct rrecord * reinforcementrec);
void initrepreinforcement(struct rrecord * reinforcementrec);
int   criterion(struct rrecord * rrec, struct erecord * environrec);
void payreward(struct poptype * population, struct rrecord * rrec, struct crecord * clearingrec);
void reportreinforcement(struct rrecord * reinforcementrec);
void reinforcement(struct rrecord * reinforcementrec, struct poptype * population, struct crecord * clearingrec, struct erecord *
                    environrec);
void advance(struct crecord * clearingrec);
void initga(struct grecord * garec, struct poptype * population);
void initrepga(struct grecord * garec);
int   select(struct poptype * population);
int   mutation(int positionvalue, float pmutation, long &nmutation );
int   bmutation(int positionvalue, float pmutation, long &nmutation );
void crossover(struct classtype * parent1, struct classtype * parent2, struct classtype * child1, struct classtype * child2, float
                pcrossover, float pmutation, int &sitecross, int nposition, long &ncrossover, long &nmutation);
int   worstofn(struct poptype * population,  int n);
int   matchcount(struct classtype * classifier1, struct classtype * classifier2, int nposition);
int   crowding(struct classtype * child, struct poptype * population, int &crowdingfactor, int crowdingsubpop);
void statistics(struct poptype * population);
void ga(struct grecord * garec, struct poptype * population);
void reportga(struct grecord * garec, struct poptype * population );
float rndreal(float lo , float hi);
void initrandomnormaldeviate();
double randomnormaldeviate();
float noise(float mu , float sigma);
void skip(int skipcount);
int   flip(float prob);
int   rnd(int low, int high);
float fmax(float x, float y);
float fmin(float x, float y);
float avg(float x, float y);
int   halt();
void initmalloc();
void freeall();
```

```
void initrepheader()
{
  rep<<"--------------------------------------------------"<<endl;
  rep<<"                基本遗传学习分类系统               "<<endl;
  rep<<" A Simple Classifier System based on Genetic Learning "<<endl;
  rep<<"--------------------------------------------------"<<endl;

}
void interactiveheader()
{
  cout<<"--------------------------------------------------"<<endl;
  cout<<"                基本遗传学习分类系统               "<<endl;
  cout<<" A Simple Classifier System based on Genetic Learning "<<endl;
  cout<<"--------------------------------------------------"<<endl;

}

void initialization()  /* 分类系统初始化 */
{
interactiveheader();
initrandomnormaldeviate();
initmalloc();
initrepheader();
initclassifiers(pop);
initrepclassifiers(pop);
initenvironment(&environrec);
initrepenvironment(&environrec);
initaoc(&clearingrec);
initrepaoc(&clearingrec);
initreinforcement(&reinforcementrec);
initrepreinforcement(&reinforcementrec);
inittimekeeper(&timekeeprec);
initreptimekeeper(&timekeeprec);
initga(&garec, pop);
initrepga(&garec);
}

void detectors(struct erecord *environrec, int *envmessage)  /* 检测器 */
{
int j;
for(j=0;j<=environrec->lsignal-1; j++)
  envmessage[j]=environrec->signal[j];
}

void writemessage(int *mess, int lmessage)
{
  int j;
  j=lmessage-1;
    while(j>=0) {
            rep<<mess[j];
          j--;
    }
}

void reportdetectors(int *envmessage, int nposition)  /* 输出环境消息 */
{
  skip(1);
```

```
  rep<<"环境消息:      ";
  writemessage(envmessage, nposition);
  skip(1);
}

void reportheader()
{
  skip(1);
  rep<<"        详细报告 "<<endl;
  rep<<"-----------------"<<endl;

}

void report()
{
  reportheader();
  reporttime(&timekeeprec);
  reportenvironment(&environrec);
  reportdetectors(envmessage, pop->nposition);
  reportclassifiers(pop);
  reportaoc(&clearingrec);
  reportreinforcement(&reinforcementrec);
}

void consolereport(struct rrecord * reinforcementrec)
{
    cout<<"------------------------------------"<<endl;
    cout<<"    次数 = "<<reinforcementrec->totalcount<<endl;
    cout<<"    P    = "<<reinforcementrec->proportionreward<<endl;
    cout<<"    P50  = "<<reinforcementrec->proportionreward50<<endl;
    cout<<"------------------------------------"<<endl;
}

void plotreport(struct rrecord * reinforcementrec)
{
  pfile<<setw(12)<<reinforcementrec->totalcount<<setw(12)<<reinforcementrec->proportionreward<<
          setw(10)<<reinforcementrec->proportionreward50<<endl;
}

int addtime(int t, int dt, int &carryflag)
{
  int tempadd;
  tempadd = t + dt;
  carryflag = (tempadd> = iterationsperblock);
  if(carryflag)
    tempadd = (int)fmod((double)tempadd, (double)iterationsperblock);
  return(tempadd);
}

void inittimekeeper(struct trecord * timekeeprec)
{
  int dummyflag;
    timekeeprec->iteration = 0; timekeeprec->block = 0;
    tfile>>timekeeprec->initialiteration;
    tfile>>timekeeprec->initialblock;
    tfile>>timekeeprec->reportperiod;
    tfile>>timekeeprec->consolereportperiod;
    tfile>>timekeeprec->plotreportperiod;
```

```
    tfile>>timekeeprec->gaperiod;
    timekeeprec->iteration=timekeeprec->initialiteration;
    timekeeprec->block=timekeeprec->initialblock;
    timekeeprec->nextga=addtime(timekeeprec->iteration, timekeeprec->gaperiod, dummyflag);
    timekeeprec->nextreport=addtime(timekeeprec->iteration, timekeeprec->reportperiod, dummyflag);
    timekeeprec->nextconsolereport=addtime(timekeeprec->iteration, timekeeprec->consolereportperiod, dummyflag);
    timekeeprec->nextplotreport=addtime(timekeeprec->iteration, timekeeprec->plotreportperiod, dummyflag);
}

void initreptimekeeper(struct trecord * timekeeprec)
{
  skip(1);
  rep<<"        时间参数:     "<<endl;
  rep<<"-----------------------------------"<<endl;
  rep<<" 初始次数          = "<<timekeeprec->initialiteration<<endl;
  rep<<" 初始时间块        = "<<timekeeprec->initialblock<<endl;
  rep<<" 结果数据输出周期= "<<timekeeprec->reportperiod<<endl;
  rep<<" 数据屏幕显示周期= "<<timekeeprec->consolereportperiod<<endl;
  rep<<" 绘图数据输出周期= "<<timekeeprec->plotreportperiod<<endl;
  rep<<" 遗传算法执行周期= "<<timekeeprec->gaperiod<<endl;
}

void timekeeper(struct trecord * timekeeprec)
{
  int carryflag, dummyflag;
  timekeeprec->iteration=addtime(timekeeprec->iteration, 1, carryflag);
  if(timekeeprec->iteration>=iterationsperblock) timekeeprec->block=timekeeprec->block+1;
  timekeeprec->reportflag=(timekeeprec->nextreport==timekeeprec->iteration);
  if(timekeeprec->reportflag)
  timekeeprec->nextreport=addtime(timekeeprec->iteration, timekeeprec->reportperiod, carryflag);
  timekeeprec->consolereportflag=(timekeeprec->nextconsolereport==timekeeprec->iteration);
  if(timekeeprec->consolereportflag)
  timekeeprec->nextconsolereport=addtime(timekeeprec->iteration, timekeeprec->consolereportperiod, dummyflag);
  timekeeprec->plotreportflag=(timekeeprec->nextplotreport==timekeeprec->iteration);
  if(timekeeprec->plotreportflag)
  timekeeprec->nextplotreport=addtime(timekeeprec->iteration, timekeeprec->plotreportperiod, dummyflag);
  timekeeprec->gaflag=(timekeeprec->nextga==timekeeprec->iteration);
  if(timekeeprec->gaflag)
  timekeeprec->nextga=addtime(timekeeprec->iteration, timekeeprec->gaperiod, dummyflag);
}

void reporttime(struct trecord * timekeeprec)
{
  rep<<" [块:次数] = ["<< timekeeprec->block<<":"<<timekeeprec->iteration<<"]"<<endl;
}

void generatesignal(struct erecord * environrec)
{
int j;
for(j=0;j<=environrec->lsignal-1; j++)
  {
    if(flip(0.5))
        environrec->signal[j]=1;
    else
        environrec->signal[j]=0;
  }
}
```

```
int decode(int *mess, int start, int length)
{
int j, accum, powerof2;
accum = 0; powerof2 = 1;
for(j = start; j <= start + length - 1; j++)
  {
     accum = accum + powerof2 * mess[j];
     powerof2 = powerof2 * 2;
  }
return(accum);
}

void multiplexeroutput(struct erecord *environrec)
{

  environrec->address = decode(environrec->signal, 0, environrec->laddress);
  environrec->output = environrec->signal[environrec->laddress + environrec->address];
}

void environment(struct erecord *environrec)
{
generatesignal(environrec);
multiplexeroutput(environrec);
}

void initenvironment(struct erecord *environrec)
{
int j;
efile>>environrec->laddress;
environrec->ldata = (int)ceil(pow(2.0, (double)environrec->laddress));
environrec->lsignal = environrec->laddress + environrec->ldata;
environrec->address = 0;
environrec->output = 0;
environrec->classifieroutput = 0;
for(j = 0; j <= environrec->lsignal - 1; j++)
  environrec->signal[j] = 1;
}

void initrepenvironment(struct erecord *environrec)
{
  skip(1);
  rep<<"  环境参数(多路复用器):    "<<endl;
  rep<<"-------------------------------"<<endl;
  rep<<" 地址行数 = "<< environrec->laddress<<endl;
  rep<<" 数据行数 = "<< environrec->ldata<<endl;
  rep<<" 总行数   = "<< environrec->lsignal<<endl;
}

void writesignal(int *signal, int lsignal)
{
  int j;
  j = lsignal - 1;
    while(j >= 0) {
          rep<<signal[j];
        j--;
    }
}
```

```
void reportenvironment(struct erecord * environrec)
{
  skip(1);
  rep<<"        当前多路复用器状态:      "<<endl;
  rep<<"----------------------------------"<<endl;
  rep<<" 信号:  ";
  writesignal(environrec->signal, environrec->lsignal); skip(1);
  rep<<" 解码地址          = "<<environrec->address<<endl;
  rep<<" 多路复用器输出 = "<<environrec->output<<endl;
  rep<<" 分类器输出      = "<<environrec->classifieroutput<<endl;
}

int randomchar(double pgeneral)
{
  if(flip((float)pgeneral))
    return(wildcard);
  else if(flip(0.5))
    return(1);
  else
    return(0);
}

void readcondition(int * c, float pgeneral, int nposition)
{
  char ch;
  int j;
      for(j=0;j<=nposition-1;j++)
    {
    cfile>>ch;
    switch(ch)
    {
    case '0':
        c[nposition-1-j]=0;
        break;
    case '1':
        c[nposition-1-j]=1;
        break;
    case '#':
        c[nposition-1-j]=wildcard;
        break;
    case 'R':
        c[nposition-1-j]=randomchar(pgeneral);
        break;
    }
    }
}

void readclassifier(struct classtype * Class, float pgeneral, int nposition)
{
    char ch;
    readcondition(Class->c, pgeneral, nposition);
    cfile>>ch;
    cfile>>Class->a;
    cfile>>Class->strength;
    Class->bid=0.0; Class->ebid=0.0; Class->matchflag=0;
}

int counterspecificity(int * c, int nposition)
```

```
}
  int j;
  int temp;
  temp=0;
  j=0;
  while(j<=nposition-1)
  {
    if(c[j]!=wildcard) temp=temp+1;
    j++;
  }
  return(temp);
}

void initclassifiers(struct poptype *population)
{
    int j;
    cfile>>population->pgeneral;
    cfile>>population->cbid;
    cfile>>population->bidsigma;
    cfile>>population->bidtax;
    cfile>>population->lifetax;
    cfile>>population->bid1;
    cfile>>population->bid2;
    cfile>>population->ebid1;
    cfile>>population->ebid2;
  for(j=0;j<=population->nclassifier-1;j++)
  {
    readclassifier(&population->classifier[j],population->pgeneral,population->nposition);
    population->classifier[j].specificity=counterspecificity(population->classifier[j].c,population->nposition);
  }
}

void initrepclassifiers(struct poptype *population)
{
  skip(1);
  rep<<"          分类器表参数:                    "<<endl;
  rep<<"-----------------------------------------------------"<<endl;
  rep<<"分类器数目                    ="<<population->nclassifier<<endl;
  rep<<"分类器前件码长                ="<<population->nposition<<endl;
  rep<<"投标系数                      ="<<population->cbid<<endl;
  rep<<"有效投标中随机噪声的均方差    ="<<population->bidsigma<<endl;
  rep<<"投标税                        ="<<population->bidtax<<endl;
  rep<<"存活税                        ="<<population->lifetax<<endl;
  rep<<"随机概率                      ="<<population->pgeneral<<endl;
  rep<<"投标确定性基数                ="<<population->bid1<<endl;
  rep<<"投标确定性倍率                ="<<population->bid2<<endl;
  rep<<"有效投标确定性基数            ="<<population->ebid1<<endl;
  rep<<"有效投标确定性倍率            ="<<population->ebid2<<endl;
}

void writecondition(int *c,int nposition)
{
int j;
for(j=0;j<=nposition-1;j++)
  {
    switch(c[nposition-1-j])
    {
    case 1:
```

```
        rep<<"1";
        break;
      case 0:
        rep<<"0";
        break;
      case wildcard:
        rep<<"#";
        break;
      }
   }
}

void writeclassifier(struct classtype * Class, int number, int nposition)
{
   rep<<setw(4)<<number<<"   ";
   rep<<setiosflags(ios::fixed);
   rep<<setprecision(4)<<Class->strength<<"   ";
   rep<<setprecision(3)<<Class->bid;
   rep<<setw(12)<<Class->ebid;
   if(Class->matchflag)
   rep<<"      X      ";
   else
   rep<<"             ";
   writecondition(Class->c, nposition);
   rep<<"["<<setw(3)<<Class->a<<"]"<<endl;
}

void reportclassifiers(struct poptype * population)
{
   int j;
   skip(1);
   rep<<"  编号     权值     投标     有效投标     匹配     分类器  "<<endl;
   rep<<"-----------------------------------------"<<endl;
     for(j=0;j<=population->nclassifier-1; j++)
     writeclassifier(&population->classifier[j], j, population->nposition);
}

int match(int * c, int * m, int nposition)
{
   int matchtemp, np;
   np=nposition-1;
   matchtemp=1;
   while(matchtemp && (np>=0))
   {
     matchtemp=(c[np]==wildcard) || (c[np]==m[np]);
     np=np-1;
   }
return(matchtemp);
}

void matchclassifiers(struct poptype * population, int * emess, struct classlist * matchlist)
{
   int j;
   matchlist->nactive=0;
   for(j=0;j<=population->nclassifier-1; j++)
   {
      population->classifier[j].matchflag=match(population->classifier[j].c, emess, population->nposition);
      if(population->classifier[j].matchflag)
```

```
    {
    matchlist->clist[matchlist->nactive]=j;
    matchlist->nactive=matchlist->nactive+1;
    }
  }
}

void initaoc(struct crecord *clearingrec)
{
    char ch;
    cfile>>ch;
    clearingrec->bucketbrigadeflag=(ch=='y')||(ch=='Y');
    clearingrec->winner=0; clearingrec->oldwinner=0;
}

void initrepaoc(struct crecord *clearingrec)
{
  skip(1);
  rep<<"        信任分配参数:  "<<endl;
  rep<<"-------------------------------"<<endl;
  rep<<"  桶队列标志变量   =";
  if(clearingrec->bucketbrigadeflag)
  rep<<"1"<<endl;
  else
  rep<<"0"<<endl;
}

int auction(struct poptype *population, struct classlist *matchlist, int oldwinner)
{
int j, k, winner;
float bidfmaximum;
bidfmaximum=0.0;
winner=oldwinner;
if(matchlist->nactive>=0)
  {
    for(j=0;j<=matchlist->nactive-1;j++)
      {
      k=matchlist->clist[j];
      population->classifier[k].bid=population->cbid*(population->bid1+
        population->bid2*population->classifier[k].specificity)*population->classifier[k].strength;
      population->classifier[k].ebid=population->cbid*(population->ebid1+
          population->ebid2*population->classifier[k].specificity)*population->classifier[k].strength+noise(0, pop-
ulation->bidsigma);
      if(population->classifier[k].ebid>bidfmaximum)
        {
        winner=k;
        bidfmaximum=population->classifier[k].ebid;
        }
      }
    }
  return(winner);
}

void clearinghouse(struct poptype *population, struct crecord *clearingrec)
{
  float payment;
  payment=population->classifier[clearingrec->winner].bid;
  population->classifier[clearingrec->winner].strength=population->classifier[clearingrec->winner].strength-pay-
```

```
ment;
  if(clearingrec->bucketbrigadeflag)
  population->classifier[clearingrec->oldwinner].strength = population->classifier[clearingrec->oldwinner].strength +
payment;
}

void taxcollector(struct poptype *population)
{
  int j;
  float bidtaxswitch;
  if((population->lifetax!=0.0)||(population->bidtax!=0.0))
  {
    for(j=0;j<=population->nclassifier-1;j++)
    {
        if(population->classifier[j].matchflag)
          bidtaxswitch=1.0;
        else
          bidtaxswitch=0.0;
        population->classifier[j].strength=population->classifier[j].strength-
              population->lifetax*population->classifier[j].strength-
              population->bidtax*bidtaxswitch*population->classifier[j].strength;
        }
    }
}

void reportaoc(struct crecord *clearingrec)
{
  skip(1);
  rep<<" 新胜者 ["<<clearingrec->winner<<"]  旧胜者 ["<<clearingrec->oldwinner<<"]"<<endl;
}

void aoc(struct poptype *population, struct classlist *matchlist, struct crecord *clearingrec)
{
    clearingrec->winner=auction(population, matchlist, clearingrec->oldwinner);
    taxcollector(population);
    clearinghouse(population, clearingrec);
}

void initreinforcement(struct rrecord *reinforcementrec)
{
  rfile>>reinforcementrec->reward;
  reinforcementrec->rewardcount=0.0;
  reinforcementrec->rewardcount50=0.0;
  reinforcementrec->totalcount=0.0;
  reinforcementrec->count50=0.0;
  reinforcementrec->proportionreward=0.0;
  reinforcementrec->proportionreward50=0.0;
  reinforcementrec->lastwinner=0;
}

void initrepreinforcement(struct rrecord *reinforcementrec)
{
  skip(1);
  rep<<"        增强参数：  "<<endl;
  rep<<"-----------------------------------"<<endl;
  rep<<" 增强得分 ="<<reinforcementrec->reward<<endl;
}
```

```
int criterion(struct rrecord * rrec, struct erecord * environrec)
{
    int tempflag;
    tempflag=(environrec->output==environrec->classifieroutput);
    rrec->totalcount=rrec->totalcount+1;
    rrec->count50=rrec->count50+1;
    if(tempflag)
    {
      rrec->rewardcount=rrec->rewardcount+1;
      rrec->rewardcount50=rrec->rewardcount50+1;
    }
    rrec->proportionreward=rrec->rewardcount/rrec->totalcount;
    if((int)ceil(rrec->count50-50.0)==0)
    {
    rrec->proportionreward50=(float)(rrec->rewardcount50/50.0);
    rrec->rewardcount50=0.0; rrec->count50=0.0;
    }
    return(tempflag);
}

void payreward(struct poptype * population, struct rrecord * rrec, struct crecord * clearingrec)
{
  population->classifier[clearingrec->winner].strength=
        population->classifier[clearingrec->winner].strength+rrec->reward;
  rrec->lastwinner=clearingrec->winner;
}

void reportreinforcement(struct rrecord * reinforcementrec)
{
  skip(1);
  rep<<"          增强报告:"<<endl;
  rep<<"--------------------------------------------------"<<endl;
  rep<<" 正确比例(from start) ="<<reinforcementrec->proportionreward<<endl;
  rep<<" 正确比例(last fifty) ="<<reinforcementrec->proportionreward50<<endl;
  rep<<" 最终获胜分类器号     ="<<reinforcementrec->lastwinner<<endl;
}

void reinforcement(struct rrecord * reinforcementrec, struct poptype * population, struct crecord * clearingrec, struct erecord *
                    environrec)
{
  if(criterion(reinforcementrec, environrec))
    payreward(population, reinforcementrec, clearingrec);
}

void advance(struct crecord * clearingrec)
{
  clearingrec->oldwinner=clearingrec->winner;
}

void initga(struct grecord * garec, struct poptype * population)
{
    gfile>>garec->populationselect;
    gfile>>garec->pmutation;
    gfile>>garec->pcrossover;
    gfile>>garec->crowdingfactor;
    gfile>>garec->crowdingsubpop;
    garec->nselect=(int)(garec->populationselect * population->nclassifier * 0.5);
    garec->nmutation=0; garec->ncrossover=0;
```

```
}

void initrepga(struct grecord * garec)
{
  skip(1);
  rep<<"          遗传算法参数："<<endl;
  rep<<"-----------------------------------------------------"<<endl;
  rep<<" 选择分类器比例  ="<<garec->populationselect<<endl;
  rep<<" 选择分类器数目  ="<<garec->nselect<<endl;
  rep<<" 变异概率        ="<<garec->pmutation<<endl;
  rep<<" 交叉概率        ="<<garec->pcrossover<<endl;
  rep<<" 排挤因子        ="<<garec->crowdingfactor<<endl;
  rep<<" 排挤子种群      ="<<garec->crowdingsubpop<<endl;
}

int select(struct poptype * population)
{
double random;
float partsum;
int j;
partsum=0.0; j=-1;
random=rand()/32767. * population->sumstrength;
do
  {
  j++;
  partsum=partsum+population->classifier[j].strength;
  } while((partsum<random) &&(j<population->nclassifier));
  return(j);
}

int mutation(int positionvalue, float pmutation, long &nmutation )
{
int tempmutation;
if(flip(pmutation))
  {
    tempmutation=(int)fmod( (double)(positionvalue+rnd(1,2)+1), 3.0)-1;
    nmutation=nmutation+1;
  }
else
    tempmutation=positionvalue;
return( tempmutation);
}

int bmutation(int positionvalue, float pmutation, long &nmutation )
{
int tempmutation;
if(flip(pmutation))
  {
    tempmutation=(int)fmod( (double)(positionvalue+1), 2.0);
    nmutation=nmutation+1;
  }
else
    tempmutation=positionvalue;
return( tempmutation);
}

void crossover(struct classtype * parent1, struct classtype * parent2, struct classtype * child1, struct classtype * child2, float
          pcrossover, float pmutation, int &sitecross, int nposition, long &ncrossover, long &nmutation)
```

```
{
float inheritance;
int j;
if(flip(pcrossover))
{
   sitecross = rnd(0, nposition - 2);
   ncrossover = ncrossover + 1;
}
else
   sitecross = nposition;
   child1 - >a = bmutation(parent1 - >a, pmutation, nmutation);
   child2 - >a = bmutation(parent2 - >a, pmutation, nmutation);
j = sitecross;
while(j< = nposition - 1)
{
   child2 - >c[j] = mutation(parent1 - >c[j], pmutation, nmutation);
   child1 - >c[j] = mutation(parent2 - >c[j], pmutation, nmutation);
   j = j + 1;
}
j = 0;
while(j<sitecross)
{
   child1 - >c[j] = mutation(parent1 - >c[j], pmutation, nmutation);
   child2 - >c[j] = mutation(parent2 - >c[j], pmutation, nmutation);
   j = j + 1;
}
inheritance = avg(parent1 - >strength, parent2 - >strength);
child1 - >strength = inheritance; child1 - >matchflag = 0;
child1 - >ebid = 0.0; child1 - >bid = 0.0;
child1 - >specificity = counterspecificity(child1 - >c, nposition);
child2 - >strength = inheritance; child2 - >matchflag = 0;
child2 - >ebid = 0.0; child2 - >bid = 0.0;
child2 - >specificity = counterspecificity(child2 - >c, nposition);
}

int worstofn(struct poptype * population, int n)
{
int j, worst, candidate;
float worststrength;
worst = rnd(0, population - >nclassifier - 1);
worststrength = population - >classifier[worst].strength;
if(n>0)
{
      for(j = 1;j< = n - 1;j + + )
   {
   candidate = rnd(0, population - >nclassifier - 1);
   if(worststrength>population - >classifier[candidate].strength)
      {
      worst = candidate;
      worststrength = population - >classifier[worst].strength;
      }
   }
}
return(worst);
}

int matchcount(struct classtype * classifier1, struct classtype * classifier2, int nposition)
{
```

```
int tempcount, j;
    if(classifier1->a==classifier2->a)
        tempcount=1;
    else
        tempcount=0;
    for(j=0;j<=nposition-1;j++)
        if( classifier1->c[j]==classifier2->c[j])    tempcount=tempcount+1;
    return(tempcount);
}

int crowding(struct classtype *child, struct poptype *population, int &crowdingfactor, int crowdingsubpop)
{
int popmember, j, match, matchfmax, mostsimilar;
matchfmax=-1; mostsimilar=0;
if(crowdingfactor<1) crowdingfactor=1;
for(j=0;j<=crowdingfactor-1;j++)
{
    popmember=worstofn(population, crowdingsubpop);
    match=matchcount(child, &population->classifier[popmember], population->nposition);
    if(match>matchfmax)
    {
        matchfmax=match;
        mostsimilar=popmember;
    }
  }
return(mostsimilar);
}

void statistics(struct poptype *population)
{
  int j;
  population->fmaxstrength=population->classifier[0].strength;
  population->fminstrength=population->classifier[0].strength;
  population->sumstrength=population->classifier[0].strength;
j=1;
while(j<=population->nclassifier-1)
  {
  population->fmaxstrength=fmax(population->fmaxstrength, population->classifier[j].strength);
  population->fminstrength=fmin(population->fminstrength, population->classifier[j].strength);
  population->sumstrength=population->sumstrength+population->classifier[j].strength;
  j++;
  }
  population->avgstrength=population->sumstrength/population->nclassifier;
}

void ga(struct grecord *garec, struct poptype *population)
{
int j;
struct classtype *child;
int nbytes;
child=(struct classtype *)malloc(2*sizeof(struct classtype));
nbytes=population->nposition*sizeof(int);
child[0].c=(int *) malloc(nbytes);
child[1].c=(int *) malloc(nbytes);

statistics(population);
for(j=0;j<=garec->nselect-1;j++)
  {
```

```
    garec->mating[j].mate1=select(population);
    garec->mating[j].mate2=select(population);
    crossover(&population->classifier[garec->mating[j].mate1],&population->classifier[garec->mating[j].mate1],
            &child[0],&child[1], garec->pcrossover, garec->pmutation, garec->mating[j].sitecross, population->
            nposition, garec->ncrossover, garec->nmutation);
    garec->mating[j].mort1=crowding(&child[0], population, garec->crowdingfactor, garec->crowdingsubpop);
    population->sumstrength=population->sumstrength-population->classifier[garec->mating[j].mort1].strength+
child[0].strength;
    population->classifier[garec->mating[j].mort1]=child[0];
    garec->mating[j].mort2=crowding(&child[1], population, garec->crowdingfactor, garec->crowdingsubpop);
    population->sumstrength=population->sumstrength-population->classifier[garec->mating[j].mort2].strength+
child[1].strength;
    population->classifier[garec->mating[j].mort2]=child[1];
  }
}

void reportga(struct grecord *garec, struct poptype *population )
{
    int j;
  skip(1);
  rep<<"              遗传算法报告: "<<endl;
  rep<<"-----------------------------------------------"<<endl;
  rep<<"对数    父1    父2    交叉位置    子1    子2  "<<endl;
  rep<<"-----------------------------------------------"<<endl;
    for(j=0; j<=garec->nselect-1;j++)
    {
    rep<<setw(6)<<j<<setw(6)<< garec->mating[j].mate1;
    rep<<setw(6)<<garec->mating[j].mate2<<setw(6)<<garec->mating[j].sitecross;
    rep<<setw(6)<<garec->mating[j].mort1<<setw(6)<<garec->mating[j].mort2<<endl;
    }
  rep<<"统计数据:"<<endl;
  rep<<"-----------------------"<<endl;
  rep<<"平均权值="<<population->avgstrength<<endl;
  rep<<"最大权值="<<population->fmaxstrength<<endl;
  rep<<"最小权值="<<population->fminstrength<<endl;
  rep<<"权值和  ="<<population->sumstrength<<endl;
  rep<<"交叉数目="<<garec->ncrossover<<endl;
  rep<<"变异数目="<<garec->nmutation<<endl;
}

float rndreal(float lo , float hi)  /* 在浮点数 lo 和 hi 之间产生一个随机实数 */
{
    return((float)(rand()/32767. * (hi - lo)) + lo);
}

void initrandomnormaldeviate()  /* 随机标准差的初始化 */
{
    rndcalcflag = 1;
}

double randomnormaldeviate()  /* 产生随机标准差 */
{
    double t, rndx1;
    if(rndcalcflag)
    {    rndx1 = sqrt(- 2.0 * log((double) rand()/32767 ));
         t = 6.2831853072 * (double) rand()/32767.;
         rndx2 = rndx1 * sin(t);
         rndcalcflag = 0;
```

```
            return(rndx1 * cos(t));
    }
    else
    {
        rndcalcflag = 1;
        return(rndx2);
    }
}


float noise(float mu, float sigma)  /* 产生具有给定均值和标准均方差的随机数 */
{
    return((float)(randomnormaldeviate() * sigma) + mu);
}

void skip(int skipcount)
{
    int j;
    for (j = 1; j <= skipcount; j++) rep<<endl;
}

int flip(float prob)  /* 以一定概率产生 0 或 1 */
{
    if(prob==1.0)
      return(1);
    else
      return((rand()/32767. <= prob));
}

int rnd(int low, int high)  /* 在整数 low 和 high 之间产生一个随机整数 */
{
    int i;
      if(low >= high)
        i = low;
    else
    {
        i = (int)(rand()/32767. * (high - low + 1)) + low;
        if(i > high) i = high;
    }
    return(i);
}


float fmax(float x, float y)
{
if(x>y)
return(x);
else
return(y);
}

float fmin(float x, float y)
{
if(x<y)
return(x);
else
return(y);
}
```

```
float avg(float x, float y)
{
return((float)(0.5 * (x + y)));
}

int halt()
{
const times = 1000;
int temp;
int j;
/* int ch; */
j = 0;
  do
  {
    j++;
  }while(j < times);
  temp = (j < times);
  /*
  cout << "是否继续? (1/0)";
  cin >> ch;
  if(ch == 1)
    return(0);
  else
    return(1);  */
  return(temp);
}

void initmalloc()  /* 为全局数据变量分配空间 */
{
  int nbytes, ncbytes;
  int j;
  int fmaxc, fmaxp;
  cfile >> fmaxp;
  cfile >> fmaxc;
  ncbytes = fmaxc * (fmaxp * sizeof(int) + 3 * sizeof(int) + 3 * sizeof(float));
    if((pop -> classifier = (struct classtype * ) malloc(ncbytes)) == NULL)
      { cout << "malloc: out of memory making !!" << endl;
        exit(-1);
      }
  nbytes = fmaxp * sizeof(int);
  for(j = 0; j <= fmaxc - 1; j++)
  {
    if((pop -> classifier[j].c = (int * ) malloc(nbytes)) == NULL)
      { cout << "malloc: out of memory making !!" << endl;
        exit(-1);
      }
  }
  nbytes = fmaxp * sizeof(int);
    if((envmessage = (int * ) malloc(nbytes)) == NULL)
      { cout << "malloc: out of memory making !!  \n" << endl;
        exit(-1);
      }
    if((matchl -> clist = (int * ) malloc(pop -> nclassifier * sizeof(int))) == NULL)
      { cout << "malloc: out of memory making !!  \n" << endl;
        exit(-1);
      }
```

```
  nbytes = fmaxp * sizeof(int);
    if((environrec.signal = (int *) malloc(nbytes)) == NULL)
      { cout<<"malloc: out of memory making !!  \n"<<endl;
        exit(-1);
      }
  nbytes = fmaxmating * sizeof(struct mrecord);
    if((garec.mating = (struct mrecord *) malloc(nbytes)) == NULL)
      { cout<<"malloc: out of memory making !!  \n"<<endl;
        exit(-1);
      }
  pop->nclassifier = fmaxc;
  pop->nposition = fmaxp;
}

void generatecfile()
{
    int nclassifier, nposition;
    float pgeneral, cbid, bidsigma, bidtax, lifetax, bid1, bid2, ebid1, ebid2;
    float strength;
    int i, j;
    float temprand;
    ofstream rcfile("c:\\scs\\cfile.txt");

    cout<<" 分类器前件码长 = ";
    cin>>nposition;
    rcfile<<nposition<<endl;

    cout<<" 分类器数目 = ";
    cin>>nclassifier;
    rcfile<<nclassifier<<endl;

    cout<<" 随机概率 = ";
    cin>>pgeneral;
    rcfile<<pgeneral<<endl;

    cout<<" 投标系数 = ";
    cin>>cbid;
    rcfile<<cbid<<endl;

    cout<<" 有效投标中随机噪声的均方差 = ";
    cin>>bidsigma;
    rcfile<<bidsigma<<endl;

    cout<<" 投标税 = ";
    cin>>bidtax;
    rcfile<<bidtax<<endl;

    cout<<" 存活税 = ";
    cin>>lifetax;
    rcfile<<lifetax<<endl;

    cout<<" 投标确定性基数 = ";
    cin>>bid1;
    rcfile<<bid1<<endl;

    cout<<" 投标确定性倍率 = ";
    cin>>bid2;
    rcfile<<bid2<<endl;
```

```
    cout<<" 有效投标确定性基数 = ";
    cin>>ebid1;
    rcfile<<ebid1<<endl;

    cout<<" 有效投标确定性倍率 = ";
    cin>>ebid2;
    rcfile<<ebid2<<endl;

    cout<<" 分类器初始权值 = ";
    cin>>strength;

  for(i=0;i<=nclassifier-1;i++)
  {
      for(j=0;j<=nposition-1;j++)
      {
        temprand=(float)(rand()/32767.);
      if(temprand<1./3)
        rcfile<<"0";
      else if((temprand>=1/3.)&&(temprand<2/3.))
        rcfile<<"1";
      else
        rcfile<<"#";
      }
        rcfile<<":";
        temprand=(float)(rand()/32767.);
      if(temprand<1/2.)
        rcfile<<"0";
      else
        rcfile<<"1";
      rcfile<<" "<<strength<<endl;
    }
    rcfile<<"n"<<endl;
}

void freeall()  /* 释放内存空间 */
{
int j;
free(pop->classifier);
for(j=0;j<=pop->nclassifier-1;j++)
  free(pop->classifier[j].c);
free(matchl->clist);
free(envmessage);
free(environrec.signal);
free(garec.mating);
    }

void main()  /* 主程序 */
{
/* generatecfile(); */
initialization();
detectors(&environrec,envmessage);
report();
do {
     timekeeper(&timekeeprec);
     environment(&environrec);
     detectors(&environrec,envmessage);
       matchl->nactive=0;
```

```
      matchclassifiers(pop, envmessage, matchl);
      aoc(pop, matchl, &clearingrec);
   environrec.classifieroutput = pop->classifier[clearingrec.winner].a;
   reinforcement(&reinforcementrec, pop, &clearingrec, &environrec);
   if(timekeeprec.reportflag)    report();
   if(timekeeprec.consolereportflag)    consolereport(&reinforcementrec);
   if(timekeeprec.plotreportflag)    plotreport(&reinforcementrec);
   advance(&clearingrec);
   if(timekeeprec.gaflag)
      {
      ga(&garec, pop);
      if(timekeeprec.reportflag)    reportga(&garec, pop);
      }
   }while(halt()==0);
   report();
   freeall();
}
```

Ⅱ.3 遗传优化神经网络源程序

```
/*****************************************************************/
/*                 遗传优化神经网络结构 GA_NN.C                   */
/*   A Gentic-based Optimization of the XOR Neural Network Structure   */
/*           同济大学计算机系   王小平     2000年5月               */
/*****************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<graphics.h>
#include<math.h>
#include<time.h>
#include<string.h>
#include "graph.c"
/* 宏定义 */
#define POP_SIZE        25      /* 种群大小 */
#define S_RATE          0.4     /* 选择操作时的淘汰率 */
#define M_RATE          0.01    /* 变异率 */
#define I_MAX           10      /* 输入层最大结点数 */
#define H_MAX           40      /* 隐层最大结点数 */
#define O_MAX           10      /* 输出层最大结点数 */
#define P_MAX           10      /* 最大训练样本数 */
#define MAX_G_LENGTH    200     /* 最大染色体长度 */
#define GX1             360     /* 遗传算法适应度图形区窗口的左上角点X坐标 */
#define GY1             66      /* 遗传算法适应度图形区窗口的左上角点Y坐标 */
#define GX2             360     /* 随机搜索法适应度图形区窗口的左上角点X坐标 */
#define GY2             257     /* 随机搜索法适应度图形区窗口的左上角点Y坐标 */
#define GXR             250     /* 适应度图形区窗口长度 */
#define GYR             100     /* 适应度图形区窗口宽度 */
#define GSTEP           2       /* 适应度图形X方向步长 */
/* 全局变量 */
unsigned char gene[POP_SIZE][MAX_G_LENGTH];           /* 当前世代的个体遗传基因 */
unsigned char i_unit[I_MAX],h_unit[H_MAX],o_unit[O_MAX]; /* 输入层、隐层、输出层结点值 */
unsigned char p_i_unit[P_MAX][I_MAX];                 /* 处理输入信号的值 */
unsigned char p_o_unit[P_MAX][O_MAX];                 /* 处理输出信号的值 */
double fitness[POP_SIZE];                             /* 当前世代的个体适应度 */
double max_fit,avg_fit;                               /* 当前世代的最大适应度和平均适应度 */
int  i_num,h_num,o_num,t_num;                         /* 各层结点数 */
int  p_num;                                           /* 训练样本数 */
int  g_length;                                        /* 染色体长度 */

void initialize_gene(gene,pop_size,g_length)
/* 种群中个体遗传基因型的初始化 */
unsigned char *gene;   /* 遗传基因 */
int pop_size;          /* 种群大小 */
int g_length;          /* 个体染色体长度 */
{
    int i,j;
    randomize();
    for(i=0;i<pop_size;i++)
    for(j=0;j<g_length;j++)
        *(gene+i*g_length+j)=random(2);
    }

long int calc_g_length()
/* 计算染色体长度 */
```

```
{
    int i;
    long int num;
    num = 0;
    for(i = 1; i <= h_num; i++)
      num = num + i_num + (i - 1);
      num = num + o_num * (i_num + h_num);
return(num);
}

void uni_crossover(gene, g1, g2, g3, ratio1, g_length)   /* 均匀交叉 */
unsigned char *gene;          /* 遗传基因 */
int g1, g2, g3;               /* g1 g2 父个体编号 g3 子个体编号 */
double ratio1;                /* 父个体 g1 被选中的概率 */
int g_length;                 /* 个体染色体长度 */
{
    unsigned char *gene1; /* 父 1 遗传基因的指针 */
    unsigned char *gene2; /* 父 2 遗传基因的指针 */
    unsigned char *gene3; /* 子遗传基因的指针 */
    double rnd, r1;
    int i;
    gene1 = gene + g_length * g1;
    gene2 = gene + g_length * g2;
    gene3 = gene + g_length * g3;
    r1 = (int)(10000.0 * ratio1);
    for(i = 0; i < g_length; i++)
    {     rnd = random(10000);
        if(rnd <= r1)        *(gene3 + i) = *(gene1 + i);
        else                 *(gene3 + i) = *(gene2 + i);
    }

void set_network()
/* 输入神经网络的结点数 */
{
  int i, ok;
  long int work;
  char choice[2];
  settextstyle(0, 0, 4);
  gprintf(215, 15, 4, 0, "GA-NN");
  setcolor(9);
  disp_hz24("遗传优化神经网络结构", 170, 50, 25);
  disp_hz16("==========确定网络结点数==========", 10, 150, 20);
    ok = 1;
    while(ok == 1)
      {
      setcolor(9);
      disp_hz16("输入层结点数:", 10, 180, 20);
      gscanf(300, 180, 9, 15, 3, "%s", choice);
      i_num = atoi(choice);
      setcolor(9);
      disp_hz16("隐层结点数:", 10, 210, 20);
      gscanf(300, 210, 15, 15, 3, "%s", choice);
      h_num = atoi(choice);
      setcolor(9);
      disp_hz16("输出层结点数:", 10, 240, 20);
      gscanf(300, 240, 15, 0, 4, "%s", choice);
      o_num = atoi(choice);
      t_num = i_num + h_num + o_num;
```

```
        work=calc_g_length();
        if(work>MAX_G_LENGTH)
          disp_hz16("结点数太多,请重输入。",10,250,20);
        else
          {
          g_length=(int)work;
          setcolor(9);
          disp_hz16("ok? 1:no, 2:yes",10,270,20);
          gscanf(300,270,4,0,4,"%s",choice);
          ok=atoi(choice);
          }
        }
}

void set_problem()
/* 读入训练样本 */
{
  int i,j;
  FILE *fopen(),*fpt;
  char st[100];
  setcolor(9);
  disp_hz16("训练样本文件名:",10,300,20);
  gscanf(300,300,4,0,20,"%s",st);
  if((fpt=fopen(st,"r"))==NULL) exit(-1);
  else
    {
    fscanf(fpt,"%d",&p_num);
    for(i=1;i<=p_num;i++)
      {
        for(j=1;j<=i_num;j++)
          fscanf(fpt,"%d",&p_i_unit[i-1][j-1]);
        for(j=1;j<=o_num;j++)
          fscanf(fpt,"%d",&p_o_unit[i-1][j-1]);
      }
    fclose(fpt);
    }
}

void g_draw_frame(x1,y1,x2,y2,c1,c2,c3,c4,text)
/* 在图形区指定位置写文本 */
int x1,y1,x2,y2,c1,c2,c3,c4;
/* (x1,y1)为左上角坐标,(x2,y2)为右下角坐标,c1 为背景色,c2 外框颜色,c3 为文本背景色,c4 为文本前景色 */
char *text;
{  int n,x3;
   g_rectangle(x1,y1,x2,y2,c1,1);
   g_rectangle(x1,y1,x2,y2,c2,0);
   g_rectangle(x1,y1,x2,y1+16,c3,1);
   g_rectangle(x1,y1,x2,y1+16,c2,0);
   n=strlen(text);
   x3=x1+((x2-x1-n*8)/2);
   disp_hz16(text,x3,y1,c4);
}

void g_init_frames()
/* 初始化画面中的窗口 */
{
   int i,j,cx,cy,x,y;
```

```
  char text[17];
  g_draw_frame(0,0,639,399,15,0,4,15,
      "遗传优化神经网络结构");
  g_draw_frame(0,16,320,170,7,15,8,15,"训练样本");
  for(i=1;i<=p_num;i++)
  {
    y=48+(i-1)*16;
    setcolor(9);
    disp_hz16("No  :",20,y,15);
    itoa(i,text,10);
    g_text(48,y+8,4,text);
    x=76;
    for(j=1;j<=i_num;j++)
      {
      itoa(p_i_unit[i-1][j-1],text,3);
      g_text(x,y+8,4,text);
      x=x+8;
      }
    setcolor(4);
    disp_hz16("===>",x,y,15);
    x=x+30;
    for(j=1;j<=o_num;j++)
      {
      itoa(p_o_unit[i-1][j-1],text,3);
      g_text(x,y+8,4,text);
      x=x+16;
      }
    }
  g_draw_frame(0,170,320,399,7,15,8,15,"最大适应度对应的神经元连接矩阵");
  g_draw_frame(320,16,639,207,7,15,8,15,"遗传算法适应度曲线");
  g_draw_frame(320,207,639,399,7,15,8,15,"随机搜索适应度曲线");
  }

void g_init_graphs()
/* 适应度图形窗口的初始化 */
{
  g_rectangle(GX1,GY1,GX1+GXR,GY1+GYR,0,1);
  g_rectangle(GX1,GY1,GX1+GXR,GY1+GYR,6,0);
  setcolor(1);
  disp_hz16("最大适应度",GX1+5,GY1-18,15);
  g_line(GX1+90,GY1-10,GX1+110,GY1-10,1);
  setcolor(4);
  disp_hz16("平均适应度",GX1+120,GY1-18,15);
  g_line(GX1+205,GY1-10,GX1+225,GY1-10,4);
  setcolor(15);
  disp_hz16("世代数",GX1+168,GY1+GYR+10,15);
  g_text(GX1-20,GY1,15,"1.0");
  g_text(GX1-20,GY1+GYR,15,"0.0");

  g_rectangle(GX2,GY2,GX2+GXR,GY2+GYR,0,1);
  g_rectangle(GX2,GY2,GX2+GXR,GY2+GYR,6,0);
  setcolor(1);
  disp_hz16("最大适应度",GX2+5,GY2-18,15);
  g_line(GX2+90,GY2-10,GX2+110,GY2-10,1);
  setcolor(4);
  disp_hz16("平均适应度",GX2+120,GY2-18,15);
  g_line(GX2+205,GY2-10,GX2+225,GY2-10,4);
```

```
  setcolor(15);
  disp_hz16("世代数",GX2+168,GY2+GYR+10,15);
  g_text(GX2-20,GY2,15,"1.0");
  g_text(GX2-20,GY2+GYR,15,"0.0");
}

void swap_fit(n1,n2)
/* 编号为 n1 和 n2 的个体互换 */
int n1,n2;
{
    unsigned char c;
    double f;
    int i;
/* 基因型互换 */
  for(i=0;i<g_length;i++)
    {
    c=gene[n1][i];
    gene[n1][i]=gene[n2][i];
    gene[n2][i]=c;
    }
/* 适应度互换 */
f=fitness[n1];
fitness[n1]=fitness[n2];
fitness[n2]=f;
}

void sort_fitness(p_size)
/* 种群中个体按适应度排序 */
int p_size;
{
  int i,j;
  for(i=0;i<p_size-1;i++)
  for(j=i+1;j<p_size;j++)
    if(fitness[j]>fitness[i]) swap_fit(j,i);
  max_fit=fitness[0];  /* 最大适应度计算 */
  /* 平均适应度计算 */
  avg_fit=0;
    for(i=0;i<p_size;i++)
  avg_fit=avg_fit+fitness[i]/(double)p_size;
}

void make_offspring(g1,g2,g3,ratio)
/* 用均匀交叉和变异产生后代个体，g1 和 g2 为父个体，g3 为子个体。
int g1,g2,g3;  /* 个体编号 */
double ratio;  /* 从父个体 1 的选择基因的概率 */
{
  int i,rnd;
  uni_crossover(gene,g1,g2,g3,ratio,g_length);
  for(i=0;i<g_length;i++)
   {
    rnd=random(100);
    /* 遗传基因 0,1,2 等概率选择 */
    if(rnd<=32)          gene[g3][i]=0;
    else if (rnd<=65)    gene[g3][i]=1;
    else                 gene[g3][i]=2;
   }
 }
```

```
void ga_reproduction()
/* 产生新一代种群 */
{
  int i, n, p1, p2;
  n=(int)(POP_SIZE * S_RATE);
  for(i=0;i<n;i++)
  {
  p1=random(n);
  p2=random(n);
  while(p2==p1)   p2=random(n);
  make_offspring(p1, p2, POP_SIZE-i-1, 0.5);
  }
}

void g_disp_fitness(n, gen_num, mfold, afold, mf, af)
/* 更新适应度曲线 */
  int n, gen_num;
  double mfold, afold, mf, af;
  {
    int x, y, gx, gy, x_old, y_old;
    char text[8];
    if(n==1) {gx=GX1;gy=GY1;}
      else {gx=GX2;gy=GY2;}
    if(gen_num%10==0)  /* 每隔 10 代更新一次 */
    {
        x=gx+(gen_num-1) * GSTEP;
        g_line(x, gy+1, x, gy+GYR-1, 1);
        sprintf(text, "%d", gen_num);
        if(gen_num<100||gen_num%20==0)
        g_text(x-8, gy+GYR+7, 15, text);
    }
    x_old=(int)(gx+(gen_num-1) * GSTEP);
    x=x_old+(int)GSTEP;
    y_old=(int)gy+GYR-mfold * GYR;
    y=(int)gy+GYR-mf * GYR;
    g_line(x_old, y_old, x, y, 1);
    y_old=gy+GYR-(int)(afold * GYR);
    y=gy+GYR-(int)(af * GYR);
    g_line(x_old, y_old, x, y, 4);
}

void calc_fitness(p_size)
/* 计算种群中所有个体的适应度 */
int p_size;
{
  int i, j, k, m, counter, sum;
  int gnum, correct, weight;
  double d1, d2;
  for(i=0;i<p_size;i++)
{
  counter=0;   /* 正确输出信号次数的记数 */
  for(j=0;j<p_num;j++)
  {
    /* 输入信号处理 */
    for(k=0;k<i_num;k++)
    i_unit[k]=p_i_unit[j][k];
    /* 隐层结点状态初始化为 0 */
```

```
  for(k=0;k<h_num;k++)
  h_unit[k]=0;
  /* 隐层结点状态转换 */
  gnum=0;
  for(k=0;k<h_num;k++)
  {
  sum=0;
  for(m=0;m<i_num;m++)
    {
    weight=gene[i][gnum];
    if(weight==2) weight=-1;
    sum=sum+weight*i_unit[m];
    gnum++;
    }
  if(k>0)
  {
  for(m=0;m<k;m++)
    {
    weight=gene[i][gnum];
    if(weight==2) weight=-1;
    sum=sum+weight*h_unit[m];
    gnum++;
    }
  }
  if(sum>0) h_unit[k]=1;
    else h_unit[k]=0;
  }
/* 输出结点状态初始化为 0 */
for(k=0;k<o_num;k++)
o_unit[k]=0;
/* 输出层结点状态转换 */
for(k=0;k<o_num;k++)
  {
    sum=0;
    for(m=0;m<i_num;m++)
    {
      weight=gene[i][gnum];
      if(weight==2) weight=-1;
      sum=sum+weight*i_unit[m];
      gnum++;
    }
    for(m=0;m<h_num;m++)
    {
      weight=gene[i][gnum];
      if(weight==2) weight=-1;
      sum=sum+weight*h_unit[m];
      gnum++;
    }
      if(sum>0)   o_unit[k]=1;
      else        o_unit[k]=0;
  }
  /* 统计正确输出信号的记数 */
  correct=1;
  for(k=0;k<o_num;k++)
  if(o_unit[k]!=p_o_unit[j][k]) correct=0;
  counter=correct+counter;
```

```
    d1 = (double)p_num;
    d2 = (double)counter;
    fitness[i] = d2/d1;
  }
}

void g_disp_char(x, y, x1, y1, x2, y2, v)
/* 在指定位置(x, y)写一字符变量 */
int x, y, x1, y1, x2, y2;   /* (x1, y1)为左上角坐标, (x2, y2)为右下角坐标 */
unsigned char v;
{
   char c[10];
   if(x>=x1&& x<=x2-10 && y>=y1 && y<=y2-10)
   {
   switch(v)
     {
       case 0: strcpy(c, "0\0");break;
       case 1: strcpy(c, "+\0");break;
       case 2: strcpy(c, "-\0");break;
       case 3: strcpy(c, "x\0");
     }
     g_text(x, y, 4, c);
     }
}

void g_disp_best_indv()
/* 显示最大适应度个体对应的神经网络连接矩阵 */
{
     int x1, y1, x2, y2, x, y;
     int i, j, n, g_num;
   x1=40;y1=190;x2=304;y2=384;
   g_rectangle(x1, y1, x2, y2, 7, 1);
   y=y1+16;g_num=0;
     for(i=0;i<t_num;i++)
   {
     x=x1;
     for(j=0;j<t_num;j++)
       {
         if((i>=i_num && i<i_num+h_num && j<i) ||
         (i>=i_num+h_num && j<i_num+h_num))
       { g_disp_char(x, y, x1, y1, x2, y2, gene[0][g_num]);
         g_num++;
       }
       else g_disp_char(x, y, x1, y1, x2, y2, 3);
     x=x+10;
     }
     y=y+10;
   }
}

void make_random_gene(p_size)
/* 随机产生初始种群的遗传编码 */
{
   int i, j, rnd;
   for(i=0;i<p_size;i++)
   for(j=0;j<g_length;j++)
   {
     rnd=random(100);
```

```
    /* 遗传基因 0,1,2 按等概率发生 */
    if(rnd<=32)          gene[i][j]=0;  /* 与神经元连接描述 0 对应,图中显示为"0" */
      else if(rnd<=65)   gene[i][j]=1;  /* 与神经元连接描述 1 对应,图中显示为"+" */
      else               gene[i][j]=2;  /* 与神经元连接描述 -1 对应,图中显示为"-" */
  }
}

void ga_search()
/* 遗传搜索算法 */
{
  int gen_num;
  double mfold, afold;
  make_random_gene(POP_SIZE);
  /* 适应度计算与排序 */
  calc_fitness(POP_SIZE);
  sort_fitness(POP_SIZE);
  /* 显示最大适应度个体对应的神经网络连接矩阵 */
  g_disp_best_indv();
  mfold=max_fit;afold=avg_fit;
  ga_reproduction();
  for(gen_num=1;gen_num<=120;gen_num++)
  {
  calc_fitness(POP_SIZE);
  sort_fitness(POP_SIZE);
  g_disp_best_indv();
  /* 更新适应度曲线 */
  g_disp_fitness(1, gen_num, mfold, afold, max_fit, avg_fit);
  mfold=max_fit;afold=avg_fit;
  /* 产生新一代种群 */
  ga_reproduction();
  getch();
  }
}

void random_search()
/* 随机搜索算法 */
{
  int p_size, gen_num;
  double mfold, afold;
  p_size=(int)(POP_SIZE*S_RATE);
  make_random_gene(p_size);
  calc_fitness(p_size);
  sort_fitness(p_size);
  mfold=max_fit;afold=avg_fit;
  for(gen_num=1;gen_num<=120;gen_num++)
  {
  make_random_gene(p_size);
  calc_fitness(p_size);
  sort_fitness(p_size);
  g_disp_fitness(2, gen_num, mfold, afold, max_fit, avg_fit);
  mfold=max_fit;afold=avg_fit;
  }
}

main() /* 主程序 */
{
randomize();
/* 图形界面初始化 */
```

```
g_init();
setcolor(5);
setbkcolor(15);
/* 神经网络结点数设置 */
set_network();
/* 输入训练样本 */
set_problem();
/* 初始化图形区 */
g_init_frames();
g_init_graphs();
/* 遗传搜索 */
ga_search();
/* 随机搜索 */
random_search();
disp_hz16("回车键结束",10,430,20);
getch();
}
```

Ⅱ.4 遗传识别提取基元源程序

```
/***********************************************************************/
/*              基于遗传算法的基元识别与提取 PATMAT.C                    */
/*        Genetic-based Pattern Matching and Primitive Extraction       */
/*              同济大学计算机系  王小平     2000 年 5 月                 */
/***********************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<graphics.h>
#include<math.h>
#include<time.h>
#include<string.h>
#include "graph.c"
/* 宏定义 */
#define POP_SIZE     20     /* 种群大小 */
#define S_RATE       0.4    /* 选择操作时淘汰率设定 */
#define M_RATE       0.2    /* 变异概率 */
#define DEFOCUS_L    4      /* 基元轮廓平滑处理的灰阶数 */
#define G_LENGTH     15     /* 个体染色体长度 */
#define XUL          100    /* 原图像窗口左上角点的 X 坐标 */
#define YUL          250    /* 原图像窗口左上角点的 Y 坐标 */
#define GX1          360    /* 遗传算法适应度图形窗口的左上角点 X 坐标 */
#define GY1          66     /* 遗传算法适应度图形窗口的左上角点 Y 坐标 */
#define GX2          360    /* 随机算法适应度图形窗口的左上角点 X 坐标 */
#define GY2          257    /* 随机算法适应度图形窗口的左上角点 Y 坐标 */
#define GXR          250    /* 适应度图形窗口长度 */
#define GYR          100    /* 适应度图形窗口宽度 */
#define GSTEP        2      /* 适应度图形 X 方向步长 */
/* 全局变量 */
unsigned  char  gene[POP_SIZE][G_LENGTH];  /* 当前世代个体的遗传基因 */
unsigned  char  image[128][128];           /* 原二值图像数据 */
double  fitness[POP_SIZE];                 /* 当前世代个体的适应度 */
double  _sin[8];_cos[8];                   /* 45 度三角函数表 */
double  max_fit, avg_fit;                  /* 当前世代最大适应度与平均适应度 */
int  m_num;                                /* 原图像构成像素点数 */
int  m_pnts[2][1000];                      /* 图像中像素点位置坐标 */
int  pnts[2][1000];                        /* 在图形窗口显示用的像素点位置坐标 */

void initialize_gene(gene, pop_size, g_length)
/* 种群中个体遗传基因型的初始化 */
unsigned char *gene;        /* 遗传基因 */
int pop_size;               /* 种群大小 */
int g_length;               /* 个体染色体长度 */
{
    int i,j;
    randomize();
    for(i=0;i<pop_size;i++)
    for(j=0;j<g_length;j++)
        *(gene+i*g_length+j)=random(2);
    }

void two_crossover(gene, g1, g2, g3, g4, g_length)    /* 两点交叉 */
unsigned char *gene;                                  /* 遗传基因排列 */
int g1, g2, g3, g4;                                   /* g1 g2 父个体编号 g3 g4 子个体编号 */
int g_length;                                         /* 个体遗传基因的位长 */
```

```
{
    unsigned char *gene1;        /* 父1遗传基因的指针 */
    unsigned char *gene2;        /* 父2遗传基因的指针 */
    unsigned char *gene3;        /* 子1遗传基因的指针 */
    unsigned char *gene4;        /* 子2遗传基因的指针 */
    int c_pos1, c_pos2;
    int i, j;
    int work;
    double r;
    gene1 = gene + g_length * g1;
    gene2 = gene + g_length * g2;
    gene3 = gene + g_length * g3;
    gene4 = gene + g_length * g4;
    c_pos1 = random(g_length - 1);
    c_pos2 = random(g_length - 1);
    while(c_pos1 == c_pos2)
        c_pos2 = random(g_length - 1);
    if(c_pos1 > c_pos2)
    {
    work = c_pos1; c_pos1 = c_pos2; c_pos2 = work;
    }
    for(j = 0; j < g_length; j++)
      {
    if(j >= c_pos1 && j <= c_pos2)
         {
          *(gene3 + j) = *(gene2 + j);
          *(gene4 + j) = *(gene1 + j);
         }
       else
         {
          *(gene3 + j) = *(gene1 + j);
          *(gene3 + j) = *(gene2 + j);
    }
  }
}

void g_draw_frame(x1, y1, x2, y2, c1, c2, c3, c4, text)
/* 在图形区指定位置写文本 */
int x1, y1, x2, y2, c1, c2, c3, c4;
/* (x1, y1)为左上角坐标,(x2, y2)为右下角坐标,c1 为背景色,c2 外框颜色,c3 为文本背景色,c4 为文本前景色 */
char *text;
{   int n, x3;
    g_rectangle(x1, y1, x2, y2, c1, 1);
    g_rectangle(x1, y1, x2, y2, c2, 0);
    g_rectangle(x1, y1, x2, y1 + 16, c3, 1);
    g_rectangle(x1, y1, x2, y1 + 16, c2, 0);
    n = strlen(text);
    x3 = x1 + ((x2 - x1 - n * 8)/2);
    disp_hz16(text, x3, y1, c4);
}

void g_disp_image()
/* 显示原图像 */
{
     int i, j;
     for(i = 0; i < 128; i++)
        for(j = 0; j < 128; j++)
```

```
        g_pset(XUL + j, YUL + i, image[j][i]);
}

void g_init_frames()
/* 初始化画面 */
{
   int i, j, cx, cy, x, y;
   char text[17];
   g_draw_frame(0, 0, 639, 399, 15, 0, 4, 15,
      "基于遗传算法的基元识别与提取");
   g_draw_frame(0, 16, 320, 170, 7, 15, 8, 15, "基元");
   itoa(m_num, text, 10);
   setcolor(9);
   disp_hz16("构成点数 = ", 30, 50, 15);
   disp_hz16(text, 118, 50, 15);
   for(i = 0; i < m_num; i++)
      g_pset(m_pnts[0][i] + 160, m_pnts[1][i] + 110, 4);
   g_draw_frame(0, 170, 320, 399, 7, 15, 8, 15, "");
   g_draw_frame(0, 170, 320, 399, 7, 15, 8, 15, "原图像");
   g_rectangle(XUL - 1, YUL - 1, XUL + 128, YUL + 128, 15, 0);
   g_disp_image();
   setcolor(9);
   disp_hz16("像素数 = 128X128", 30, 200, 15);
   disp_hz16("灰阶数 = ", 30, 220, 15);
   itoa(DEFOCUS_L, text, 10);
   disp_hz16(text, 102, 220, 15);
   g_draw_frame(320, 16, 639, 207, 7, 15, 8, 15, "");
   g_draw_frame(320, 16, 639, 207, 7, 15, 8, 15, "遗传算法适应度进化曲线");
   g_draw_frame(320, 207, 639, 399, 7, 15, 8, 15, "随机搜索适应度变化曲线");
   }

void g_init_graphs()
/* 适应度图形窗口初始化 */
{
   g_rectangle(GX1, GY1, GX1 + GXR, GY1 + GYR, 0, 1);
   g_rectangle(GX1, GY1, GX1 + GXR, GY1 + GYR, 6, 0);
   setcolor(1);
   disp_hz16("最大适应度", GX1 + 5, GY1 - 18, 15);
   g_line(GX1 + 90, GY1 - 10, GX1 + 110, GY1 - 10, 1);
   setcolor(4);
   disp_hz16("平均适应度", GX1 + 120, GY1 - 18, 15);
   g_line(GX1 + 205, GY1 - 10, GX1 + 225, GY1 - 10, 4);
   setcolor(15);
   disp_hz16("世代数", GX1 + 168, GY1 + GYR + 10, 15);
   g_text(GX1 - 20, GY1, 15, "1.0");
   g_text(GX1 - 20, GY1 + GYR, 15, "0.0");

   g_rectangle(GX2, GY2, GX2 + GXR, GY2 + GYR, 0, 1);
   g_rectangle(GX2, GY2, GX2 + GXR, GY2 + GYR, 6, 0);
   setcolor(1);
   disp_hz16("最大适应度", GX2 + 5, GY2 - 18, 15);
   g_line(GX2 + 90, GY2 - 10, GX2 + 110, GY2 - 10, 1);
   setcolor(4);
   disp_hz16("平均适应度", GX2 + 120, GY2 - 18, 15 );
   g_line(GX2 + 205, GY2 - 10, GX2 + 225, GY2 - 10, 4);
   setcolor(15);
   disp_hz16("世代数", GX2 + 168, GY2 + GYR + 10, 15);
```

```
  g_text(GX2-20, GY2, 15, "1.0");
  g_text(GX2-20, GY2+GYR, 15, "0.0");
}

void g_set_model()
/* 设置基元识别和提取的模型 */
{
  int n;
  int x1, y1, x2, y2, x3, y3;
  int i, j, ok, n1, n2, yn, col;
  double rad[90], wx, wy, work;
  char choice[2];
  ok=1;
  while(ok==1)
  {
    g_cls();
    settextstyle(0, 0, 4);
    gprintf(220, 20, 4, 0, "PM-GA");
    setcolor(9);
    disp_hz24("基于遗传算法的基元识别与提取", 150, 50, 25);
    setcolor(9);
    disp_hz16("选择元图形:", 10, 150, 20);
    disp_hz16("1:三角形,2:矩形 3:圆或椭圆,4:不规则形==>", 10, 300, 20);
    gscanf(420, 300, 4, 0, 2, "%s", choice);
    n=atoi(choice);
    switch(n)
    {
      case 1:    /* 三角形 */
        x1=345;y1=225;
        x2=x1-25-random(25);y2=y1;
        x3=x1;y3=y1-25-random(25);
        g_line(x1, y1, x2, y2, 9);
        g_line(x1, y1, x3, y3, 9);
        g_line(x2, y2, x3, y3, 9);break;
      case 2:    /* 矩形 */
        x1=345;y1=225;
        x2=x1-25-random(25);
        y2=y1-25-random(25);
        g_rectangle(x2, y2, x1, y1, 9, 0);break;
      case 3:    /* 圆或椭圆 */
        x1=320;y1-200;
        x2=10+random(25);
        y2=10+random(25);
        g_circle(x1, y1, x2, y2, 9);break;
      case 4:    /* 不规则形 */
        x1=320;y1=200;
        rad[0]=10+random(15);
        for(i=1;i<90;i++)
        {
      rad[i]=rad[i-1]-4+random(9);
        if(rad[i]<10) rad[i]=10.0; else
        if(rad[i]>25) rad[i]=25.0;
        }
   /* 轮廓平滑处理 */
     for(i=1;i<3;i++)
       for(j=0;j<90;j++)
         {
```

```
        if((j-1)<0) n1=89; else n1=j-1;
        if((j+1)>89) n2=0; else n2=j+1;
        rad[j]=(rad[n1]+rad[j]+rad[n2])/3.0;
        }
      for(i=0;i<90;i++)
    {
        if((i+1)>89) n1=0; else n1=i+1;
        x2=(int)(rad[i]*cos(i/45.0*3.14))+x1;
        y2=(int)(rad[i]*sin(i/45.0*3.14))+y1;
        x3=(int)(rad[n1]*cos((double)n1/45.0*3.14))+x1;
        y3=(int)(rad[n1]*sin((double)n1/45.0*3.14))+y1;
        g_line(x2,y2,x3,y3,9);
      }
}
  g_rectangle(260,140,380,260,4,0);
  setcolor(9);
  disp_hz16("轮廓圆滑处理满意否? no:1 yes:2===>",10,340,20);
  gscanf(400,340,4,0,2,"%s",choice);
  ok=atoi(choice);
}
  setcolor(9);
  disp_hz16("加入噪声否? no:1 yes:2===>",10,380,20);
  gscanf(400,380,9,0,2,"%s",choice);
  yn=atoi(choice);
  if(yn==2)
  for(i=0;i<3000;i++)
  g_pset(261+random(119),141+random(119),0);
  m_num=0;
  for(y1=141;y1<260;y1++)
    for(x1=261;x1<380;x1++)
    {
      col=g_pget(x1,y1);
      if(col!=0)
      {
        m_pnts[0][m_num]=x1;
        m_pnts[1][m_num]=y1;
        m_num=m_num+1;
      }
    }/* 相对坐标变换处理 */
wx=0;wy=0;work=(double)m_num;
for(i=1;i<=m_num;i++)
  {
    wx=wx+(double)(m_pnts[0][i-1])/work;
    wy=wy+(double)(m_pnts[1][i-1])/work;
  }
  x1=(int)wx;y1=(int)wy;
for(i=1;i<=m_num;i++)
  {
    m_pnts[0][i-1]=m_pnts[0][i-1]-x1;
    m_pnts[1][i-1]=m_pnts[1][i-1]-y1;
  }
}

void g_set_image()
/* 加入噪声和基元轮廓平滑处理后,绘制原图像 */
{
  int ok,i;
```

```
  int cx, cy, gx, gy, x, y, col, xx, yy, xymax;
  double angle, x1, y1;
  char choice[2];
  /* 产生 45 度函数表 */
  for(i=0;i<8;i++)
  {
  _sin[i]=sin((double)i*45.0/360.0*3.141592);
  _cos[i]=cos((double)i*45.0/360.0*3.141592);
}
xymax=64;
cx=200;cy=100;
ok=1;
while(ok==1)
{
    g_cls();
    for(i=0;i<25;i++) printf("\n");
      g_rectangle(cx-1, cy-1, cx+128, cy+128, 9, 0);
    for(i=0;i<3;i++)
      g_line(random(128)+cx, random(128)+cy,
        random(128)+cx, random(128)+cy, 9);
    for(i=0;i<1;i++)
      g_circle(random(64)+cx+32, random(64)+cy+32,
        random(20)+10, random(20)+10, 9);
    for(i=0;i<5000;i++)
      g_pset(random(128)+cx, random(128)+cy, 0);
    for(i=0;i<200;i++)
      g_pset(random(128)+cx, random(128)+cy, 9);
    gx=random(xymax)-xymax/2;
    gy=random(xymax)-xymax/2;
    angle=random(8);
    for(i=1;i<=m_num;i++)
    {
      x1=(double)m_pnts[0][i-1];
      y1=(double)m_pnts[1][i-1];
      x=(int)(_cos[angle]*x1-_sin[angle]*y1)+64+cx+gx;
      y=(int)(_sin[angle]*x1+_cos[angle]*y1)+64+cy+gy;
      g_pset(x, y, 9);
    }
  setcolor(9);
  disp_hz16("绘制原图像满意否? no:1 yes:2-==>", 10, 240, 20);
  gscanf(300, 240, 4, 0, 2, "%s", choice);
  ok=atoi(choice);
}
  for(y=0;y<128;y++)
  for(x=0;x<128;x++)
    {
      col=g_pget(x+cx, y+cy);
      if(col>0) image[x][y]=DEFOCUS_L;
      else image[x][y]=0;
    }
  for(y=0;y<128;y++)
 {
 /* printf("%d/128\n", y+1); */
  for(x=0;x<128;x++)
    if(image[x][y]==DEFOCUS_L)
      for(i=0;i<DEFOCUS_L;i++)
        for(gy=-i;gy<=i;gy++)
```

```
        for(gx = -i;gx<=i;gx++)
          {
            xx=x+gx;yy=y+gy;
            if(xx>=0 && xx<=127 && yy>0 && y<=127)
            if(image[xx][yy]==(DEFOCUS_L-i))
              image[xx][yy]=DEFOCUS_L-i;
          }
  }
for(y=0;y<128;y++)
  for(x=0;x<128;x++)
    g_pset(x+cx, y+cy, image[x][y]);
setcolor(9);
disp_hz16("回车键结束",10,430,20);
getch();
for(i=0;i<25;i++) printf("\n");
}

void swap_fit(n1, n2)
/* 编号为 n1 和 n2 的个体互换 */
int n1, n2;
{
    unsigned char c;
    double f;
    int i;
    /* 遗传基因型互换 */
    for(i=0;i<G_LENGTH;i++)
    {
      c=gene[n1][i];
      gene[n1][i]=gene[n2][i];
      gene[n2][i]=c;
    }
/* 适应度互换 */
f=fitness[n1];
fitness[n1]=fitness[n2];
fitness[n2]=f;
}

void sort_fitness(p_size)
/* 种群中的适应度排序 */
int p_size;
{
  int i,j;
  for(i=0;i<p_size-1;i++)
  for(j=i+1;j<p_size;j++)
    if(fitness[j]>fitness[i]) swap_fit(j,i);
  max_fit=fitness[0];   /* 最大适应度计算 */
  /* 计算平均适应度 */
  avg_fit=0;
  for(i=0;i<p_size;i++)
  avg_fit=avg_fit+fitness[i]/(double)p_size;
}

void gene_to_param(num, xsft, ysft, angle)
/* 解码，将遗传基因型转化为图形变换参数 xsft, ysft 和 angle */
int num, * xsft, * ysft, * angle;
{
int xymax, i;
```

```
xymax=64;
*xsft=0; *ysft=0;
for(i=0;i<6;i++)
{
  *xsft= *xsft*2+gene[num][i];
  *ysft= *ysft*2+gene[num][i+6];
}
  *xsft= *xsft-xymax/2;
  *ysft= *ysft-xymax/2;
  *angle=0;
 for(i=12;i<15;i++)
  *angle= *angle*2+gene[num][i];
}

void calc_fitness(p_size)
/* 计算种群中所有个体的适应度 */
int p_size;
{
 int i,j,xsft,ysft,angle,x1,y1,x2,y2;
 double sum,fitmax;
 fitmax=(double)(DEFOCUS_L*m_num);
   for(i=0;i<p_size;i++)
   {
 gene_to_param(i,&xsft,&ysft,&angle);
 sum=0;
 for(j=0;j<m_num;j++)
   {
   x1=(double)m_pnts[0][j];
   y1=(double)m_pnts[1][j];
   x2=(int)(_cos[angle]*x1-_sin[angle]*y1)+64+xsft;
   y2=(int)(_sin[angle]*x1+_cos[angle]*y1)+64+ysft;
   if(x2>=0 && x2<128 && y2>=0 && y2<128)
       sum=sum+(double)image[x2][y2];
   }
     fitness[i]=sum/fitmax;
 }
}

void g_disp_max(flg)
/* 将提取到的当前代最大适应度个体对应的基元在原图像中重复绘制出来 */
int flg;  /* 当 flg=0 为初始代,flg=1 时为进化世代 */
{
 int i,xsft,ysft,angle,x,y;
 /* 消除前代绘制的基元 */
 if(flg==1)
 {
   for(i=0;i<m_num;i++)
 {
   x=pnts[0][i]; y=pnts[1][i];
   if(x>=0 && x<128 && y>=0 && y<128)
     g_pset(XUL+x,YUL+y,image[x][y]);
 }
}
 /* 从当前代最佳个体遗传基因型获取基元变换参数 */
 gene_to_param(0,&xsft,&ysft,&angle);
for(i=0;i<m_num;i++)
{
    x=(double)m_pnts[0][i];
```

```
    y-(double)m_pnts[1][i];
    pnts[0][i]=(int)(_cos[angle]*x-_sin[angle]*y)+64+xsft;
    pnts[1][i]=(int)(_sin[angle]*x+_cos[angle]*y)+64+ysft;
}
for(i=0;i<m_num;i++)
{
    x=pnts[0][i];
    y=pnts[1][i];
  if(x>=0 && x<128 && y>=0 && y<128)
    g_pset(XUL+x,YUL+y,9);
  }
}

void make_offspring(g1,g2,g3,g4)
/* 用两点交叉和变异产生后代个体，g1和g2为父个体，g3和g4为子个体。
int g1,g2,g3,g4;
{
  int i,rnd,rndmax;
  /* 两点交叉 */
  two_crossover(gene,g1,g2,g3,g4,G_LENGTH);
  /* 变异 */
  rndmax=(int)(10000.0*M_RATE);
  for(i=0;i<G_LENGTH;i++)
  {
    rnd=random(10000);
    if(rnd<=rndmax)
      if(gene[g3][i]==0) gene[g3][i]=1;
        else gene[g3][i]=0;
      rnd=random(10000);
    if(rnd<=rndmax)
      if(gene[g4][i]==0) gene[g4][i]=1;
        else gene[g4][i]=0;
  }
}

void ga_reproduction()
/* 产生新一代种群 */
{
  int i,n,p1,p2;
  n=(int)(POP_SIZE*S_RATE/2.0);
  for(i=0;i<n;i++)
  {
  p1=random(n*2);
  p2=random(n*2);
  while(p2==p1)
  p2=random(n*2);
  make_offspring(p1,p2,POP_SIZE-i*2-1,POP_SIZE-i*2-2);
  }
}

void g_disp_fitness(n,gen_num,mfold,afold,mf,af)
/* 随世代进化更新适应度曲线 */
int n,gen_num;
double mfold,afold,mf,af;
{
  int x,y,gx,gy,x_old,y_old;
  char text[8];
  if(n==1) {gx=GX1;gy=GY1;}
```

```
    else {gx=GX2;gy=GY2;}
  if(gen_num%10==0)
  {
    x=gx+(gen_num-1)*GSTEP;
    g_line(x,gy+1,x,gy+GYR-1,1);
    sprintf(text,"%d",gen_num);
    if(gen_num<100||gen_num%20==0)
    g_text(x-8,gy+GYR+7,15,text);
  }
  x_old=(int)(gx+(gen_num-1)*GSTEP);
  x=x_old+(int)GSTEP;
  y_old=(int)gy+GYR-mfold*GYR;
  y=(int)gy+GYR-mf*GYR;
  g_line(x_old,y_old,x,y,1);
  y_old=gy+GYR-(int)(afold*GYR);
  y=gy+GYR-(int)(af*GYR);
  g_line(x_old,y_old,x,y,4);
}

void ga_search()
/* 遗传搜索 */
{
  int gen_num;
  double mfold,afold;
  initialize_gene(gene,POP_SIZE,G_LENGTH);
  calc_fitness(POP_SIZE);
  sort_fitness(POP_SIZE);
  g_disp_max(0);
  mfold=max_fit;afold=avg_fit;
  ga_reproduction();
  for(gen_num=1;gen_num<=120;gen_num++)
  {
  /* 适应度计算和排序 */
  calc_fitness(POP_SIZE);
  sort_fitness(POP_SIZE);
  /* 更新适应度曲线 */
  g_disp_fitness(1,gen_num,mfold,afold,max_fit,avg_fit);
  /* 最大适应度对应的基元在原图像中绘制 */
  g_disp_max(1);
  mfold=max_fit;afold=avg_fit;
  /* 产生新一代种群 */
  ga_reproduction();
}
  g_disp_image();
}

void make_random_gene(p_size)
/* 随机产生初始个体的遗传基因型 */
int p_size;
{
  int i,j,rnd;
    for(i=0;i<p_size;i++)
    for(j=0;j<15;j++)
      if(random(100)<50) gene[i][j]=0;
        else gene[i][j]=1;
}
```

```
void random_search()
/* 随机搜索 */
{
  int p_size, gen_num;
  double mfold, afold;
  p_size = (int)(POP_SIZE * S_RATE);
  make_random_gene(p_size);
  calc_fitness(p_size);
  sort_fitness(p_size);
  g_disp_max(0);
  mfold = max_fit; afold = avg_fit;
  for(gen_num = 1; gen_num <= 120; gen_num++)
  {
  make_random_gene(p_size);
  calc_fitness(p_size);
  sort_fitness(p_size);
  g_disp_fitness(2, gen_num, mfold, afold, max_fit, avg_fit);
  g_disp_max(1);
  mfold = max_fit; afold = avg_fit;
  }
  g_disp_image();
}

main()  /* 主程序 */
{
randomize();
/* 图形界面初始化 */
g_init();
setcolor(9);
setbkcolor(15);
/* 设置基元识别和提取的模型 */
g_set_model();
/* 加入噪声和基元轮廓平滑处理后,绘制原图像 */
g_set_image();
/* 初始化图形设置 */
g_init_frames();
/* 适应度图形窗口初始化 */
g_init_graphs();
/* 遗传搜索 */
ga_search();
/* 随机搜索 */
random_search();
disp_hz16("回车键结束", 10, 430, 20);
getch();
}
```

Ⅱ.5 基于遗传算法的人工生命模拟源程序

```
/*****************************************************************/
/*                基于遗传算法的人工生命模拟 AI_GA.C                    */
/*      An Artificial Life Simulation model Based on Genetic Algorithm   */
/*                同济大学计算机系  王小平      2000年5月                  */
/*****************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<graphics.h>
#include<math.h>
#include<time.h>
#include<string.h>
#include "graph.c"
/* 宏定义 */
#define TL1          20     /* 植物性食物限制时间 */
#define TL2          5      /* 动物性食物限制时间 */
#define NEWFOODS     3      /* 植物性食物每代生成数目 */
#define MUTATION     0.05   /* 变异概率 */
#define G_LENGTH     32     /* 个体染色体长度 */
#define MAX_POP      100    /* 个体总数的最大值 */
#define MAX_FOOD     100    /* 食物总数的最大值 */
#define MAX_WX       60     /* 虚拟环境的长度最大值 */
#define MAX_WY       32     /* 虚拟环境的宽度最大值 */
#define SX1          330    /* 虚拟环境图左上角点x坐标 */
#define SY1          40     /* 虚拟环境图左上角点y坐标 */
#define GX           360    /* 个体数进化图形窗口的左上角点X坐标 */
#define GY           257    /* 个体数进化图形窗口的左上角点Y坐标 */
#define GXR          250    /* 个体数进化图形窗口的长度 */
#define GYR          100    /* 个体数进化图形窗口的宽度 */
#define GSTEP        2      /* 个体数进化图形窗口的X方向步长 */
#define R_LIFE       0.05   /* 初期产生生物数的环境比率 */
#define R_FOOD       0.02   /* 初期产生食物数的环境比率 */
#define SL_MIN       10     /* 个体寿命最小值 */
/* 全局变量 */
unsigned char gene[MAX_POP][G_LENGTH];      /* 遗传基因 */
unsigned char iflg[MAX_POP];                /* 个体死活状态标志变量 */
unsigned char fflg[MAX_FOOD];               /* 食物有无状态标志变量 */
unsigned char world[MAX_WX][MAX_WY];        /* 虚拟环境的数据 */
unsigned char                               /* 各中行为模式数据 */
life1[5][5]={{0,0,1,0,0},{0,1,0,1,0},{1,0,0,0,1},{0,1,0,1,0},{0,0,1,0,0}};
unsigned char
life2[5][5]={{1,1,1,1,1},{1,0,0,0,1},{1,0,0,0,1},{1,0,0,0,1},{1,1,1,1,1}};
unsigned char
food1[5][5]={{0,0,0,1,0},{0,0,1,1,0},{0,1,0,1,0},{0,0,1,1,0},{0,0,0,1,0}};
unsigned char
food2[5][5]={{0,0,0,1,0},{0,0,1,1,0},{0,1,1,1,0},{0,0,1,1,0},{0,0,0,1,0}};
int pop_size;                  /* 个体总数 */
int iatr[MAX_POP][4];          /* 个体属性 */
                               /* iatr[][0] 个体当前位置x坐标 */
                               /* iatr[][1] 个体当前位置y坐标 */
                               /* iatr[][2] 内部能量 */
                               /* iatr[][3] 年龄属性 */
int food_size;                 /* 食物总数 */
int fatr[MAX_FOOD][4];         /* 食物属性 */
                               /* fatr[][0] 食物当前位置x坐标 */
```

```
                                /* fatr[][1] 食物当前位置 y 坐标 */
                                /* fatr[][2]=0 ：植物性 =1:动物性 */
                                /* fatr[][3] 新鲜程度 */
int wx, wy;                     /* 虚拟环境的长宽度 */

void uni_crossover(gene, g1, g2, g3, ratio1, g_length)  /* 均匀交叉 */
unsigned char *gene;   /* 遗传基因 */
int g1, g2, g3;        /* g1 g2 父个体编号 g3 子个体编号 */
double ratio1;         /* 父个体 g1 被选中的概率 */
int g_length;          /* 个体遗传基因的位长 */
{
  unsigned char *gene1;        /* 父 1 遗传基因的指针 */
  unsigned char *gene2;        /* 父 2 遗传基因的指针 */
  unsigned char *gene3;        /* 子遗传基因的指针 */
  double rnd, r1;
  int i;
  gene1 = gene + g_length * g1;
  gene2 = gene + g_length * g2;
  gene3 = gene + g_length * g3;
  r1 = (int)(10000.0 * ratio1);
  for(i=0;i<g_length;i++)
  {  rnd = random(10000);
     if(rnd<=r1)   *(gene3+i) = *(gene1+i);
     else          *(gene3+i) = *(gene2+i);
  }
}


void g_disp_unit(x, y, n)
/* 绘制虚拟环境的一个单元 */
int x, y;   /* x=0,1,2...,wx-1; y=0,1,2,...,wy-1 */
int n;   /* n=0: =1: 生物 1 =2:生物 2 =3:植物性食物 =4:障碍物 =5:动物性食物 */
{
  int gx, gy, i, j;
  unsigned char col;
  gx = SX1 + 5 * x; gy = SY1 + 5 * y;
  for(i=0;i<5;i++)
    for(j=0;j<5;j++)
    { switch(n)
       {case 0: col=0;                break;
        case 1: col=life1[j][i]*2;    break;
        case 2: col=life2[j][i]*4;    break;
        case 3: col=food1[j][i]*6;    break;
        case 4: col=7;                break;
        case 5: col=food2[j][i]*11;
       }
    g_pset(gx+j, gy+i, col);
    }
}

void g_draw_world()       /* 显示虚拟环境画面 */
{
int i, j;
for(i=0;i<wy;i++)
  for(j=0;j<wx;j++)
  g_disp_unit(j, i, world[j][i]);
}
```

```
void g_draw_frame(x1, y1, x2, y2, c1, c2, c3, c4, text)
int x1, y1, x2, y2, c1, c2, c3, c4;
char * text;
{  int n, x3;
   g_rectangle(x1, y1, x2, y2, c1, 1);
   g_rectangle(x1, y1, x2, y2, c2, 0);
   g_rectangle(x1, y1, x2, y1 + 16, c3, 1);
   g_rectangle(x1, y1, x2, y1 + 16, c2, 0);
   n = strlen(text);
   x3 = x1 + ((x2 - x1 - n * 8)/2);
   disp_hz16(text, x3, y1, c4);
}

void g_init_frames()  /* 初始化画面 */
{
  int i, j, cx, cy, x, y;
  char text[17];
  g_draw_frame(0, 0, 639, 399, 15, 0, 4, 15,
    "基于遗传算法的人工生命模拟");
  g_draw_frame(0, 16, 320, 170, 7, 0, 8, 15, "设定参数");
  y = 48;
    setcolor(9);
  disp_hz16("植物食物限制时间", 16, y, 15);
  sprintf(text, "%d", TL1);
  g_text(200, y + 8, 4, text);
  y = y + 24;
    setcolor(9);
  disp_hz16("动物食物限制时间", 16, y, 15);
  sprintf(text, "%d", TL2);
  g_text(200, y + 8, 4, text);
  y = y + 24;
    setcolor(9);
  disp_hz16("植物食物每代生成个数", 16, y, 15);
  sprintf(text, "%d", NEWFOODS);
  g_text(200, y + 8, 4, text);
  y = y + 24;
    setcolor(9);
  disp_hz16("变异率", 16, y, 15);
  i = (int)(MUTATION * 100.0);
  sprintf(text, "%d", i);
  g_text(152, y + 8, 4, text);
  g_draw_frame(0, 170, 320, 399, 7, 0, 8, 15, "最佳基因型");
  x = 16; y = 194;
    setcolor(9);
  disp_hz16("SP:物种号........", x, y, 15); y = y + 16;
  disp_hz16("SL:寿命..........", x, y, 15); y = y + 16;
  disp_hz16("VF:视野..........", x, y, 15); y = y + 16;
  disp_hz16("TM:基本移动模式..", x, y, 15); y = y + 16;
  disp_hz16("CM:移动特点......", x, y, 15); y = y + 16;
  disp_hz16("LM:移动能耗......", x, y, 15); y = y + 16;
  disp_hz16("CA:行动特点......", x, y, 15); y = y + 16;
  disp_hz16("CR:善变性........", x, y, 15); y = y + 16;
  disp_hz16("SA:攻击速度......", x, y, 15); y = y + 16;
  disp_hz16("DA:防御能力......", x, y, 15); y = y + 16;
  disp_hz16("LA:攻击能耗......", x, y, 15); y = y + 16;
  disp_hz16("EF:食物吸取效率..", x, y, 15); y = y + 16;
  g_draw_frame(320, 16, 639, 207, 7, 0, 8, 15, "虚拟世界");
```

```
  g_draw_frame(320,207,639,399,7,0,8,15,"世代个体数目变化");
  }

void g_init_graph()
/* 个体数进化图初始化 */
{
  g_rectangle(GX,GY,GX+GXR,GY+GYR,0,1);
  g_rectangle(GX,GY,GX+GXR,GY+GYR,6,0);
  setcolor(1);
  disp_hz16( "生物 1",GX+5,GY-18,15);
  g_line(GX+90,GY-10,GX+110,GY-10,1);
  setcolor(4);
  disp_hz16( "生物 2",GX+120,GY-18,15);
  g_line(GX+205,GY-10,GX+225,GY-10,4);
  setcolor(0);
  disp_hz16("世代数",GX+168,GY+GYR+10,15);
  g_text(GX-25,GY,0,"100");
  g_text(GX-14,GY+GYR,0,"0");
}

void g_plot_population(gen_num,n1,n2,n1old,n2old)
int gen_num,n1,n2,n1old,n2old;
{
  int x,y,gx,gy,x_old,y_old;
  char text[8];
  if(gen_num%10==0)
  {
    x=GX+(gen_num-1)*GSTEP;
    g_line(x,GY+1,x,GY+GYR-1,1);
    sprintf(text,"%d",gen_num);
    if(gen_num<100||gen_num%20==0)
    g_text(x-8,GY+GYR+5,15,text);
  }
  x_old=GX+(gen_num-1)*GSTEP;
  x=x_old+GSTEP;
  y_old=GY+GYR-n1old;
  y=GY+GYR-n1;
  g_line(x_old,y_old,x,y,1);
  y_old=GY+GYR-n2old;
  y=GY+GYR-n2;
  g_line(x_old,y_old,x,y,4);
}

void g_disp_genotype()
/* 显示最佳个体的遗传基因型 */
{
  int i,j,n0,n1,x,y;
  unsigned char g[G_LENGTH];
  unsigned char bits[12][2]=
  { {0,0},{1,4},{5,6},{7,8},{9,11},{12,12},{13,15},
    {16,18},{19,21},{22,24},{25,27},{28,31}};
  /* 画面消除 */
  g_rectangle(200,187,319,398,7,1);
    if(pop_size!=0)
    {
      /* 获取各遗传因子 */
        for(i=0;i<G_LENGTH;i++)
```

```
        {
        n0=0;n1=0;
        for(j=0;j<pop_size;j++)
        if(gene[j][i]==0) n0++;
        else n1++;
        if(n0>=n1) g[i]=0; else g[i]=1;
        }
        x=220;
        for(i=0;i<12;i++)
        {
        y=202+i*16;
        for(j=bits[i][0];j<=bits[i][1];j++)
        if(g[j]==0)
          g_text(x+(j-bits[i][0])*16,y,4,"0");
          else
          g_text(x+(j-bits[i][0])*16,y,4,"1");
        }
    }
  }

void g_disp_char(x,y,x1,y1,x2,y2,v)
int x,y,x1,y1,x2,y2;
unsigned char v;
{
  char c[10];
  if(x>=x1&& x<=x2-8 && y>=y1 && y<=y2-10)
  {
  switch(v)
    {
      case 0: strcpy(c,"0\0");break;
      case 1: strcpy(c,"+\0");break;
      case 2: strcpy(c,"-\0");break;
      case 3: strcpy(c,"x\0");
    }
  g_text(x,y,15,c);
  }
}
void remove_life(n)
/* 消除第n个个体 */
int n;
{
  iflg[n]=0;
  world[iatr[n][0]][iatr[n][1]]=0;
  g_disp_unit(iatr[n][0],iatr[n][1],0);
  if(food_size+1<=MAX_FOOD)
  {
  food_size++;
  fatr[food_size-1][0]=iatr[n][0];
  fatr[food_size-1][1]=iatr[n][1];
  fatr[food_size-1][2]=1;
  fatr[food_size-1][3]=0;
  fflg[food_size-1]=1;
  world[iatr[n][0]][iatr[n][1]]=5;
  g_disp_unit(iatr[n][0],iatr[n][1],5);
  }
}

void remove_food(n)
```

```
/* 消除第 n 个食物 */
int n;
{
  fflg[n]=0;
  world[fatr[n][0]][fatr[n][1]]=0;
  g_disp_unit(fatr[n][0],fatr[n][1],0);
  }

void make_lives_and_foods()
/* 设置虚拟环境中生物与食物 */
{
  int x,y,i,j;
  pop_size=0;
  food_size=0;
  for(y=0;y<wy;y++)
    for(x=0;x<wx;x++)
    {
      if(world[x][y]==1||world[x][y]==2)
      {
        if(pop_size+1<=MAX_POP)
        {
        pop_size++;
        /* 生成遗传因子 */
          gene[pop_size-1][0]=world[x][y]-1;
        for(i=1;i<G_LENGTH;i++)
          gene[pop_size-1][i]=random(2);
        /* 设定属性 */
          iatr[pop_size-1][0]=x;
          iatr[pop_size-1][1]=y;
            iatr[pop_size-1][2]=70+random(30);
          iatr[pop_size-1][3]=random(SL_MIN);
          }
        }
        if(world[x][y]==3||world[x][y]==5)
        {
        if(food_size+1<=MAX_FOOD)
        {
        food_size++;
        /* 设定属性 */
        fatr[food_size-1][0]=x;
        fatr[food_size-1][1]=y;
          if(world[x][y]==3)
        fatr[food_size-1][2]=0;
        else
          fatr[food_size-1][2]=1;
        fatr[food_size-1][3]=random(TL1-1)+1;
        }
        }
    }
}

void find_empty(x,y)
/* 寻找虚拟环境中的空处,返回坐标 */
  int *x,*y;
  {
  int ok;
  ok=0;
  while(ok==0)
```

```
  {
    *x=random(wx); *y=random(wy);
    if(world[*x][*y]==0) ok=1;
    }
  }

void make_world()
/* 随机设定人工环境 */
{
int i, j, k, num, x, y;
int ok, overlap;
char choice[3];
double size;
wx=0;
setcolor(9);
while(wx<10||wx>MAX_WX)
{
    disp_hz16("虚拟环境长度(10-60)", 10, 210, 20);
    gscanf(300, 210, 4, 0, 3, "%s", choice);
    wx=atoi(choice);
  }
wy=0;
while(wy<10||wy>MAX_WY)
{
    setcolor(9);
    disp_hz16("虚拟环境宽度(10-32)", 10, 240, 20);
    gscanf(300, 240, 4, 0, 3, "%s", choice);
    wy=atoi(choice);
  }
for(i=0;i<wy;i++)
  for(j=0;j<wx;j++)
    if(i==0||i==wy-1||j==0||j==wx-1)
    world[j][i]=4;
    else world[j][i]=0;
/* 设定障碍物 */
size=(double)(wx*wy);
num=(int)(size/40.0);
if(num>MAX_POP) num=MAX_POP;
for(i=0;i<num;i++)
{
  find_empty(&x, &y);
  world[x][y]=4;
}
num=(int)(size/5.0);
if(num>MAX_FOOD) num=MAX_FOOD;
for(i=0;i<num;i++)
{
ok=0;
while(ok==0)
{
  x=random(wx);y=random(wy);
  if((world[x][y]!=4) &&
    (world[x][y-1]==4 || world[x][y+1]==4 ||
    world[x-1][y]==4 || world[x+1][y]==4))
    { world[x][y]=4;
      ok=1;
    }
```

```
  }
}
/* printf("正在整定障碍物形状\n"); */
for(y=0;y<wy;y++)
for(x=0;x<wx;x++)
  if(world[x][y]==0)
  {
  num=0;
    for(i=-1;i<=1;i++)
      for(j=-1;j<=1;j++)
        if(get_world(x+j,y+i)==4)
        num++;
        if(num>=6) world[x][y]=4;
        }
  /* 设定生物 */
  num=(int)(size*R_LIFE);
  for(i=0;i<num;i++)
  { find_empty(&x,&y);
    world[x][y]=random(2)+1;
  }
  /* 设定食物 */
  num=(int)(size*R_FOOD);
  for(i=0;i<num;i++)
    {
    find_empty(&x,&y);
    world[x][y]=3;
  }
}

void load_world_file()
/* 读取虚拟环境数据文件设定 */
{
FILE *fopen(),*fpt;
char st[100],c;
int i,j;
setcolor(9);
  disp_hz16("设定文件名:",10,210,20);
  gscanf(300,210,4,0,20,"%s",st);
/* printf("设定文件名:");
    scanf("%s",st);
    printf("%s\n",st); */
if((fpt=fopen(st,"r"))==NULL) exit(-1);
else
{
    fscanf(fpt,"%d",&wx);
  /* printf("虚拟环境长度(10-%d):",wx); */
    fscanf(fpt,"%d",&wy);
  /* printf("虚拟环境宽度(10-%d):",wy); */
    for(i=0;i<wy;i++)
      for(j=0;j<wx;j++)
      fscanf(fpt,"%d",&world[j][i]);
      fclose(fpt);
      }
}

int get_world(x,y)
/*坐标(x,y)处环境值 */
int x,y;
```

```
{
  if(x>=0 && x<wx && y>=0 && y<wy)
  return(world[x][y]);
  else
  return(-1);
}

int decode_gene(n, sb, bw)
/* 第n个个体基因型解码 */
int n, sb, bw; /* sb开始位 bw位长 */
{
  int i, sum;
  sum=0;
  for(i=sb;i<sb+bw;i++)
  sum=sum*2+gene[n][i];
  return(sum);
  }

void move_pos(n, x1, y1, x2, y2)
/* 个体n从(x1, y1)移动到(x2, y2) */
int n, x1, y1, x2, y2;
{
  int sp, loss;
  loss=decode_gene(n, 12, 1)+1;          /* 移动消耗 */
  iatr[n][2]=iatr[n][2]-loss;            /* 内部能量更新 */
  if(iatr[n][2]<=0) remove_life(n);
  else
  {
    /* 个体属性更新 */
    iatr[n][0]=x2;iatr[n][1]=y2;         /* x, y坐标更新 */
    /* 显示更新 */
    sp=gene[n][0]+1;
    g_disp_unit(x1, y1, 0);              /* 当前位置(x, y)图形消除 */
    world[x1][y1]=0;
    g_disp_unit(x2, y2, sp);             /* 新位置图形表示 */
    world[x2][y2]=sp;
  }
}

void move_randomly(n)          /* 个体n按照移动模式随机移动 */
int n;
{

/* 基本移动模式1 */
int pat1[8][2]={{1,0},{1,1},{0,1},{-1,1},
              {-1,0},{-1,-1},{0,-1},{1,-1}};
/* 基本移动模式2与3 */
int pat2_3[2][4][2]={{{1,0},{0,1},{-1,0},{0,-1}},
              {{1,1},{-1,1},{-1,-1},{1,-1}}};
int pat, x1, y1, x2, y2, rndnum;
pat=decode_gene(n,7,2);
/* pat(0,1,2,3): 表示基本移动模式 */
x1=iatr[n][0];         /* 当前x坐标 */
y1=iatr[n][1];         /* 当前y坐标 */
if(pat<=1)             /* 基本移动模式1 */
{
  rndnum=random(8);
```

```
  x2 = x1 + pat1[rndnum][0];        /* 移动目的点 x 坐标 */
  y2 = y1 + pat1[rndnum][1];        /* 移动目的点 y 坐标 */
  }
else  /* 基本移动模式 2 与 3 */
{
  rndnum = random(4);
  x2 = x1 + pat2_3[pat - 2][rndnum][0];
  y2 = y1 + pat2_3[pat - 2][rndnum][1];
  }
  if(x2 >= 0 && x2 < wx && y2 >= 0 && y2 < wy)
    if(get_world(x2, y2) == 0)
      move_pos(n, x1, y1, x2, y2);
    /* 非法目的点的场合不作移动 */
}

void move_individual(n)          /* 个体 n 移动 */
int n;
{
int cx, cy, dx, dy, sp, vf, sumx, sumy;
int i, j, a, sgn[3], num;
double vect[8][2] = {{1,0},{1,1},{0,1},{-1,1},
              {-1,0},{-1,-1},{0,-1},{1,-1}};
double vx, vy, d1, d2;
double cos, cos_max;
cx = iatr[n][0];         /* 当前 x 坐标 */
cy = iatr[n][1];         /* 当前 y 坐标 */
sp = decode_gene(n, 0, 1) + 1;   /* 生物种 1 和 2 */
for(i = 0; i < 3; i++)           /* 移动特点 CM */
{
sgn[i] = decode_gene(n, 9 + i, 1);
if(sgn[i] == 0) sgn[i] = -1;
}
sumx = 0; sumy = 0; num = 0;
vf = decode_gene(n, 5, 2) + 1;           /* 视野 */
  for(i = -vf; i <= vf; i++)
    for(j = -vf; j <= vf; j++)
    {
    if(i! = 0 || j! = 0)
    {
      a = get_world(cx + j, cy + i);
      if(a == 1 || a == 2)          /* 生物 1 和 2 */
      { num++;
      if(a == sp)           /* 同种生物 */
      {
      sumx = sumx + sgn[0] * j;
      sumy = sumy + sgn[0] * i;
      }
      else          /* 异种生物 */
      {
      sumx = sumx + sgn[1] * j;
      sumy = sumy + sgn[1] * i;
      }
    } else
    if(a == 3 || a == 5)          /* 食物 */
      {
      num++;
      sumx = sumx + sgn[2] * j;
      sumy = sumy + sgn[2] * i;
```

```
        }
      }
}
if(num!=0)           /* 视野内有其他生物和食物时 */
{
vx=(double)sumx/(double)num;
vy=(double)sumy/(double)num;
if(vx!=0||vy!=0)
{
     cos_max=-1.0;
     j=0;
     for(i=0;i<8;i++)
     {
     d1=sqrt(vx*vx+vy*vy);
     d2=sqrt(vect[i][0]*vect[i][0]+vect[i][1]*vect[i][1]);
     _cos=(vx*vect[i][0]+vy*vect[i][1])/d1/d2;
     if(_cos>cos_max)
     {
     cos_max=_cos;j=i;
     }
     }
  dx=cx+(int)vect[j][0];
  dy=cy+(int)vect[j][1];
  if(dx>=0 && dx<wx && dy>=0 && dy<wy)
     if(world[dx][dy]==0)
     move_pos(n, cx, cy, dx, dy);
  }
  else move_randomly(n);
}
else move_randomly(n);
/* 视野内有其他生物和食物时 */
}

void act1_attack(n)
/* 个体 n 攻击行动范围内的其他生物个体 */
int n;
{
  int sft[8][2]={{1,0},{1,1},{0,1},{-1,1},
              {-1,0},{-1,-1},{0,-1},{1,-1}};
  int x1, y1, x2, y2, n2;
  int found, rndnum;
  double attack1, attack2, sa1, sa2, da1, da2, rnd1, rnd2, La1, La2;
  x1=iatr[n][0];y1=iatr[n][1];
  /* 获得攻击对象的坐标(x2, y2) */
  found=0;
  while(found==0)
  {
  rndnum=random(8);
  x2=x1+sft[rndnum][0];
  y2=y1+sft[rndnum][1];
  if(get_world(x2, y2)==1||get_world(x2, y2)==2)
  found=1;
  }
  /* 检查攻击对象个体号 n2 */
  found=0;n2=0;
  while(found==0)
  {
```

```
    if(iatr[n2][0]==x2 && iatr[n2][1]==y2 && iflg[n2]==1)
    found=1; else n2++;
  }
  /* 计算双方的 Attack 量 */
  sa1=(double)decode_gene(n,19,3);
  da1=(double)decode_gene(n,22,3);
  sa2=(double)decode_gene(n2,19,3);
  da2=(double)decode_gene(n2,22,3);
  rnd1=(double)random(1001)/1000.0;
  rnd2=(double)random(1001)/1000.0;
  attack1=(double)iatr[n][2]+sa1*20.0/7.0*rnd1+da1*20.0/7.0*rnd2;
  rnd1=(double)random(1001)/1000.0;
  rnd2=(double)random(1001)/1000.0;
  attack2=(double)iatr[n2][2]+sa2*20.0/7.0*rnd1+da2*20.0/7.0*rnd2;
  /* 减少内部能量 */
  La1=decode_gene(n,25,3);
  La2=decode_gene(n2,25,3);
  rnd1=(double)random(1001)/1000.0;
  iatr[n][2]=iatr[n][2]-(int)((double)La1*rnd1);
  rnd2=(double)random(1001)/1000.0;
  iatr[n2][2]=iatr[n2][2]-(int)((double)La2*rnd2);
  if(attack1>=attack2) /* 胜者:n 败者:n2 */
    iatr[n2][2]=iatr[n2][2]-40;
  else /* 胜者:n2 败者:n */
    iatr[n][2]=iatr[n][2]-40;
  if(iatr[n][2]<=0) remove_life(n);
  if(iatr[n2][2]<=0) remove_life(n2);
  }

void act2_eat(n)   /* 个体 n 获取行动范围内的食物 */
int n;
{
  int sft[8][2]={{1,0},{1,1},{0,1},{-1,1},
             {-1,0},{-1,-1},{0,-1},{1,-1}};
  int x1,y1,x2,y2,n2,ef;
  int found,rndnum;
  x1=iatr[n][0];y1=iatr[n][1];
  /* 获取食物位置(x2,y2) */
  found=0;
  while(found==0)
  {
  rndnum=random(8);
  x2=x1+sft[rndnum][0];
  y2=y1+sft[rndnum][1];
  if(get_world(x2,y2)==3||get_world(x2,y2)==5)
  found=1;
  }
  /* 增加内部能量 */
  ef=decode_gene(n,28,4);         /* 食物吸取效率 */
  iatr[n][2]=iatr[n][2]+(int)(40.0*(50.0+(double)ef*50.0/15.0)/100.0);
  if(iatr[n][2]>100) iatr[n][2]=100;
  /* 检查食物号 n2 */
  found=0;n2=0;
  while(found==0)
  {
    if(fatr[n2][0]==x2 && fatr[n2][1]==y2 && fflg[n2]==1)
    found=1; else n2++;
  }
```

```
  remove_food(n2);
  }

void act3_makechild(n)
/* 个体n与行动范围内的其他生物个体交配产生子个体 */
int n;
{
   int i, j, k, x, y, x2, y2, found, n2, trial;
   int x3, y3;
   double rnd;
   if(pop_size+1<=MAX_POP)
   {
      x=iatr[n][0];y=iatr[n][1];
      found=0;
      while(found==0)
      {
      x2=x+random(3)-1;
      y2=y+random(3)-1;
      if(x2!=x||y2!=y)
         if(get_world(x2,y2)==gene[n][0]+1);
         found=1;
      }
      /* 检查交配对象个体号n2 */
      found=0; n2=0;
      while(found==0)
      {
        if((iatr[n2][0]==x2 || iatr[n2][1]==y2) && iflg[n2]==1)
        found=1; else n2++;
        if(n2>=pop_size-1) return;
      }
      /* 确定产生个体位置 */
      found=0;trial=0;
      while(found==0 && trial<50)
      {
      i=random(3)-1;j=random(3)-1;
      k=random(2);
      if(k==0) { x3=x+i;y3=y+j;}
      else { x3=x2+i;y3=y2+j;}
      if(get_world(x3,y3)==0) found=1;
      trial++;
      }
      if(found==1)
      {
      /* 个体n与个体n2产生子个体 */
      pop_size++;
      /* 均匀交叉 */
      uni_crossover(gene,n,n2,pop_size-1,0.5,G_LENGTH);
      /* 变异 */
      for(i=1;i<G_LENGTH;i++)
      {
         rnd=random(10001)/10000.0;
         if(rnd<=MUTATION)
           if(gene[pop_size-1][i]==1)
              gene[pop_size-1][i]=0;
           else gene[pop_size-1][i]=1;
      }
      /* 交配后父个体能量减少 */
```

```
      iatr[n][2]=iatr[n][2]-45;
      if(iatr[n][2]<=0) remove_life(n);
      iatr[n2][2]=iatr[n2][2]-45;
      if(iatr[n2][2]<=0) remove_life(n2);
      /* 子个体属性输入 */
      iatr[pop_size-1][0]=x3;
      iatr[pop_size-1][1]=y3;
      iatr[pop_size-1][2]=100;
      iatr[pop_size-1][3]=0;
      iflg[pop_size-1]=1;
      /* 子个体画面表示 */
      world[x3][y3]=gene[pop_size-1][0]+1;
      g_disp_unit(x3, y3, gene[pop_size-1][0]+1);
      }
    }
}
void act_individual(n)
/* 为行动范围内的其他生物和食物决定行动 */
int n;
{
  int i, j, k, pattern, action, cr, ca;
  int act[3];          /* act[0]:攻击 act[1]:获取食物 act[2]:交配 */
  int pat[6][3]={{1,2,3},{1,3,2},{2,1,3},
          {3,1,2},{2,3,1},{3,2,1}};
  /* pat:行动优先顺序{攻击,获取食物,交配} */
  int sp;
  double rnd;
  sp=decode_gene(n,0,1)+1;
  for(i=0;i<3;i++) act[i]=0;
  for(i=-1;i<=1;i++)
    for(j=-1;j<=1;j++)
    {
      if(i!=0||j!=0)
      {
      k=get_world(iatr[n][0]+j, iatr[n][1]+i);
      if(k==1||k==2)   act[0]=1;
      if(k==3||k==5)   act[1]=1;
      if(k==sp)        act[2]=1;
      }
      }
      cr=decode_gene(n,16,3);
      rnd=(double)random(10001)/10000.0;
      if(rnd<=(double)cr/7.0)
      {
      action=random(3);
      while(act[action]==0)
      action=random(3);
      }
      else
      {
      ca=decode_gene(n,13,3);          /* ca 行动特点 */
      if(ca<3) pattern=0;else pattern=ca-2;
      /* 基本行动模式 pattern 0-5 */
      i=0;
      action=pat[pattern][i]-1;
      while( act[action]==0)
      {
      i++;
```

```
        action=pat[pattern][i]-1;
        }
        }
        switch(action+1)
        {
        case 1: act1_attack(n);break;
        case 2: act2_eat(n);break;
        case 3: act3_makechild(n);
        }
    }

void init_flags()
/* 状态标志初始化 */
  {
  int i;
    for(i=0;i<pop_size;i++) iflg[i]=1;
    for(i=0;i<food_size;i++) fflg[i]=1;
  }

void act_lives()           /* 改变状态(移动或行动) */
{
  int i,j,k,x,y,move,a;
  for(i=0;i<pop_size;i++)
  {
    if(iflg[i]==1)
    {
      move=1;
      for(j=-1;j<=1;j++)
      for(k=-1;k<=1;k++)
    {
    if(j!=0||k!=0)
    {
      a=get_world(iatr[i][0]+k,iatr[i][1]+j);
      if(a==1||a==2||a==3||a==5)
      move=0;
    }
    }
    if(move==1)
    move_individual(i);
    else
    act_individual(i);
    }
  }
}

void increase_age()           /* 个体年龄增1 */
{
  int i,j,s;
  for(i=0;i<pop_size;i++)
  {
    if(iflg[i]==1)
    {
      j=decode_gene(i,1,4);
      s=SL_MIN+j;
      iatr[i][3]++;
      if(iatr[i][3]>s)
      remove_life(i);
```

```
    }
  }
}

void increase_frsh()          /* 食物新鲜度增 1 */
{
  int i;
  for(i=0;i<food_size;i++)
    if(fflg[i]==1)
    {
      fatr[i][3]++;
      if((fatr[i][2]==0 && fatr[i][3]>TL1)||
      (fatr[i][2]==1 && fatr[i][3]>TL2))
      remove_food(i);
    }
}

void gabage_col()             /* 死去个体及消减食物清除 */
{
int i,j;
int new_pop,new_food;
  /* 检查食物 */
new_food=0;
for(i=0;i<food_size;i++)
  if(fflg[i]==1)
  {
  new_food++;
  for(j=0;j<4;j++)
    fatr[new_food-1][j]=fatr[i][j];
  }
    food_size=new_food;
  /* 检查个体 */
  new_pop=0;
  for(i=0;i<pop_size;i++)
    if(iflg[i]==1)
    {
      new_pop++;
      /* 遗传基因复制 */
      for(j=0;j<G_LENGTH;j++)
      gene[new_pop-1][j]=gene[i][j];
      /* 属性复制 */
      for(j=0;j<4;j++)
      iatr[new_pop-1][j]=iatr[i][j];
    }
    pop_size=new_pop;
  }

 void make_foods()            /* 产生一代中植物性食物 */
  {
  int i,x,y;
  for(i=0;i<NEWFOODS;i++)
  {
    if(food_size+1<=MAX_FOOD)
      {
      food_size++;
      find_empty(&x,&y);
      fatr[food_size-1][0]=x;
```

```
        fatr[food_size-1][1]=y;
        fatr[food_size-1][2]=0;             /* 植物性 */
        fatr[food_size-1][3]=0;
        fflg[food_size-1]=1;
        world[x][y]=3;
        g_disp_unit(x,y,3);
        }
    }
}

  void calc_population(n1,n2)            /* 计算生物1和2的个体数 */
  int *n1,*n2;

     int i,p1,p2;
     p1=0;p2=0;
     if(pop_size>0)
        for(i=0;i<pop_size;i++)
        if(gene[i][0]==0) p1++; else p2++;
         *n1=p1;
         *n2=p2;
  }

main()          /* 主程序 */
{
int i,work;
int n1,n2,n1old,n2old;
char choice[2];
randomize();
/* 图形界面初始化 */
  g_init();
  settextstyle(0,0,4);
  gprintf(220,20,4,0,"ALIFE");
  setcolor(9);
  setbkcolor(15);
  disp_hz24("基于遗传算法的人工生命模拟",150,60,25);
  setcolor(9);
disp_hz16("人工环境及生物分布",10,160,20);
disp_hz16("1:随机产生 2:读文件产生 ==>",10,190,20);
gscanf(300,190,4,1,2,"%s",choice);
work=atoi(choice);
if(work==2) load_world_file();else make_world();
make_lives_and_foods();
/*状态初始化 */
init_flags();
/*计算个体数 */
calc_population(&n1old,&n2old);
/* 生成初始画面 */
g_init_frames();
/* 虚拟世界画面 */
g_draw_world();
/* 显示初始图形 */
g_init_graph();
for(i=1;i<121;i++)
{
  /* 状态初始化 */
  init_flags();
  /* 改变状态(移动或行动) */
```

```
act_lives();
/* 个体年龄增加 */
increase_age();
/* 食物新鲜度增加 */
increase_frsh();
/* 死去个体及消减食物清除 */
gabage_col();
/* 产生新的食物 */
make_foods();
/* 求生物 1 和 2 的个体数 */
calc_population(&n1, &n2);
/* 个体数变化的图形更新 */
g_plot_population(i, n1, n2, n1old, n2old);
n1old = n1; n2old = n2;
/* 显示最佳遗传基因 */
g_disp_genotype();
getch();
}
setcolor(9);
disp_hz16("回车键结束", 10, 430, 20);
getch();
}
```

Ⅱ.1～Ⅱ.5 中用到的图形函数：

```
/*****************************************************/
/*                    图形函数 graph.c                 */
/*****************************************************/
#include <bios.h>
#include <conio.h>
#include <process.h>
#include <io.h>
#include <dos.h>

void g_init()        /* 图形界面初始化 */
{  int g_driver, g_mode;
   g_driver = DETECT;
   initgraph(&g_driver, &g_mode, "");
   if(grOk != graphresult()) exit(1);
}

void g_pset(x, y, col)       /* 描点 */
int x, y, col;
{   putpixel(x, y, col);
}

void g_line(x1, y1, x2, y2, col)       /* 绘直线 */
int x1, y1, x2, y2, col;
{setcolor(col);
 setlinestyle(0, 1, 0);
 line(x1, y1, x2, y2);
}

void g_circle(cx, cy, rx, ry, col)       /* 绘圆 */
int cx, cy, rx, ry, col;
{
  setcolor(col);
  ellipse(cx, cy, 0, 360, rx, ry);
}

void g_rectangle(x1, y1, x2, y2, col, fill)       /* 绘矩形 */
int x1, y1, x2, y2, col, fill;
{
if(fill != 1)
  { setcolor(col);
  setlinestyle(0, 1, 0);
  rectangle(x1, y1, x2, y2);
  }
else
 {
 setfillstyle(1, col);
 bar(x1, y1, x2, y2);
 }
}

void g_cls()        /* 清屏 */
{
cleardevice();
}

void g_text(x, y, col, text)       /* 写文字 */
int x, y, col;
```

```
char * text;
{
setcolor(col);
settextstyle(0, 0, 1);
outtextxy(x, y, text);
}

int g_pget(x, y)   /* 取得点颜色 */
int x, y;
{ int color;
color = getpixel(x, y);
return(color);
}

void read_chb24(unsigned int hz_code, int x1, int y1, FILE * stream)
/* 读 24 点阵汉字字库 */
{
  register int i, j;
  unsigned int gjqwm;
  struct{
  unsigned int bit0:1;
  unsigned int bit1:1;
  unsigned int bit2:1;
  unsigned int bit3:1;
  unsigned int bit4:1;
  unsigned int bit5:1;
  unsigned int bit6:1;
  unsigned int bit7:1;
  }sp[72];
  union{
  struct {
  unsigned int bit0:1;
  unsigned int bit1:1;
  unsigned int bit2:1;
  unsigned int bit3:1;
  unsigned int bit4:1;
  unsigned int bit5:1;
  unsigned int bit6:1;
  unsigned int bit7:1;
  }s;
  unsigned char byte;}cc;
  long kk;
  gjqwm = hz_code - 1500;
  kk = (gjqwm/100 - 1) * 94 + (gjqwm%100);
  kk = (kk - 1) * 72;
  fseek(stream, kk, SEEK_SET);
  fread(sp, 1, 72, stream);
  for(i = 0; i < 72; i = i + 3)
  {
  for(j = 2; j >= 0; j--)
    {
    cc.s.bit7 = sp[j + i].bit0;
    cc.s.bit6 = sp[j + i].bit1;
    cc.s.bit5 = sp[j + i].bit2;
    cc.s.bit4 = sp[j + i].bit3;
    cc.s.bit3 = sp[j + i].bit4;
    cc.s.bit2 = sp[j + i].bit5;
    cc.s.bit1 = sp[j + i].bit6;
```

```
        cc.s.bit0=sp[j+i].bit7;
        setlinestyle(USERBIT_LINE, cc.byte, 1);
        line(x1, y1+23-(2-j)*8, x1, y1+23-(2-j)*8-7);
        }
        x1=x1+1;
    }
}

int disp_hz24(unsigned char *msg, int x1, int y1, int delt)          /* 显示汉字串 */
{
  FILE *stream;
  unsigned char g1, g2;
  unsigned int hz_code;
  register int i, j;
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  if((stream=fopen("\\GA\\hzk24s", "rb"))==NULL)
  {
  setlinestyle(SOLID_LINE, 0, 1);
  return 1;
  }
  for(;*msg;msg+=2)
    {
    g1=*msg-0xa0;
    g2=*(msg+1)-0xa0;
    hz_code=g1*100+g2;
    read_clib24(hz_code, x1, y1, stream);
    x1+=delt;
    }
    fclose(stream);
    setlinestyle(SOLID_LINE, 0, 1);
    return 0;
}

void read_clib16(unsigned int hz_code, int x1, int y1, FILE *stream)
/* 读16点阵汉字字库 */
{
   register int i, j;
   unsigned int gjqwm;
   unsigned char sp[32];
   unsigned long kk, k;
   gjqwm=hz_code;
   kk=(gjqwm/100-1)*94+(gjqwm%100);
   kk=(kk-1)*32;
   fseek(stream, kk, SEEK_SET);
   fread(sp, 1, 32, stream);
   for(i=0;i<32;i=i+2)
   {
     k=sp[i]*256+sp[i+1];
     setlinestyle(USERBIT_LINE, k, 1);
     line(x1+15, y1, x1, y1);
     y1++;
   }
 }

int disp_hz16(unsigned char *msg, int x1, int y1, int delt)          /* 显示汉字串 */
{
   FILE *stream;
```

```
  unsigned char g1, g2;
  unsigned int hz_code;
  register int i, j;
  char msg_no_hz[1];
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  if((stream = fopen("\\GA\\hzk16", "rb")) == NULL)
  {
    setlinestyle(SOLID_LINE, 0, 1);
    return 1;
  }
next1: for(; *msg;msg+=2)
  {
  if( *msg<0xa0)
  {
    settextstyle(0, 0, 0);
    msg_no_hz[0]= *msg;
    msg_no_hz[1]=0;
    outtextxy(x1, y1+8, msg_no_hz);
    x1+=8;
    msg++;
  }
  if( *msg<0xa0)
  {
    settextstyle(0, 0, 0);
    msg_no_hz[0]= *msg;
    msg_no_hz[1]=0;
    outtextxy(x1, y1+8, msg_no_hz);
    x1+=8;
    msg++;
    goto next1;
  }
    g1= *msg-0xa0;
    g2= *(msg+1)-0xa0;
    hz_code=g1*100+g2;
    read_chb16(hz_code, x1, y1, stream);
    x1+=delt;
  }
  fclose(stream);
  setlinestyle(SOLID_LINE, 0, 1);
  return 0;
}

int gscanf(int x, int y, int fore, int back, int length, char *fmt, ...)
{
  va_list argptr;
  char key[40];
  int xp, yp, temp;
  int i=0, xx=x, j;
  char string[10];
  char c[1];
  union inkey{
    char ch[2];
    int i;
  }cr;
  setwritemode(1);
  va_start(argptr, fmt);
  line(x, y, x, y+8);
```

```
setcolor(fore);
setfillstyle(1, back);
settextjustify(0, 2);
settextstyle(0, 0, 1);
do{
circl:
   line(x, y, x, y + 8);
   if(bioskey(1))
      {
         cc i = bioskey(0);
      }
      else
      { delay(100);
         goto circl;
      }
      line(x, y, x, y + 8);
      if(cc.ch[0])
   {
      temp = key[i] = c[0] = cc.ch[0];
      c[1] = '\0';
      switch(temp)
      {
         case 0x0d: goto next;
         case 0x1b:
            setcolor(back);
            setwritemode(0);
            line(x, y, x, y + 8);
            key[0] = '\0';
            vsscanf(key, fmt, argptr);
            va_end(argptr);
            return 1;
         case 75 * 256:
         case 8:
            if(x< = xx) x = xx;
            setfillstyle(1, back);
            bar(x, y, x 8, y + 8);
            line(x, y, x, y + 8);
            i = i - 2;
            if(i<0) i =  1;
            break;
         defaut:
            if(cc.ch[0])
            {
            line(x, y, x, y + 8);
         setfillstyle(1, back);
            bar(x, y, x + 8, y + 8);
            setcolor(fore);
            settextstyle(0, 0, 1);
         outtextxy(x, y, c);
            x + =8;
            }
         break;
         }
      i + +;
      }
   } while((i<length) && (i> = 0));
next:
            setcolor(back);
```

```
        setwritemode(0);
        line(x, y, x, y + 8);
        key[i] = '\0';
        vsscanf(key, fmt, argptr);
        va_end(argptr);
        settextstyle(1, 0, 2);
        return 0;
}
```

附录 Ⅲ

名词术语中英文对照

A

adaptive crossover	自适应交叉
Adaptive GA, AGA	自适应遗传算法
adaptive mutation	自适应变异
allele	等位基因
antibody	抗体
antigen	抗原
Ant System	蚁元系统
architecture bit string	结构位串
arithmetic crossover	算术交叉
Artificial Brain	人工脑
artificial life	人工生命
artificial neural network	人工神经网络
auction	拍卖机制
autonomous distributed system	自治分布式系统
Automatically Defined Functions, ADF	自定义函数

B

Baldwin effect	鲍德威效应
barrier function method	障碍法
bid	投标
binary valued crossover	二进制交叉
boundary mutation	边界变异
branch and bound method	分枝定界法
building block hypothesis	积木块假设
Bucket Brigade Algorithm	桶队列算法

C

cataclysmic mutation	大变异
cell	细胞
Cellular Automata, CA	元胞自动机
cellular encoding	细胞编码
character genes	符号编码基因
chromosome	染色体
classifier	分类器
classifier system	分类系统
clearing	票据交换
coarse-grained PGA	粗粒度并行遗传算法
coevolutionary GA	协同进化遗传算法
coding	编码
combinatorial optimization problem	组合最优化问题
computational intelligence	计算智能
Constrained Optimization Problems, COPs	约束最优化问题
constraints	约束条件
convex programming	凸规划
Credit Assignment Algorithm, CAA	信任分配算法
Cross generational elitist selection	跨世代精英选择法
crossover	交叉
crossover operator	交叉算子
crossover probability	交叉概率
crossover with reduced surrogate	缩小代理交叉

crowding	排挤
cut operator	切断算子
Cycle Crossover, CX	循环交叉算子
D	
decode	解码
deoxyribonucleic acid, DNA	脱氧核糖核酸
detector	检测器
discrete recombination	离散重组
disruptive selection	破坏性选择方法
double minimum spanning tree	双极小生成树法
E	
edge recombination crossover	边重组交叉
effector	作用器
elitist model	精英保留模型
elitist expected value model	精英保留期待值模型
enumerative search	枚举搜索法
epistasis	显位基因
emergent behavior	突现行为
evaluation function	评价函数
Evolutionary Algorithms, EAs	进化算法
Evolutionary Computation, EC	进化计算
Evolutionary Programming, EP	进化规划
extended linear recombination	扩展线性重组
Evoluable Hardware, EHW	进化硬件
Evolutionary Strategy, ES	进化策略
evolutionary dynamics	进化动力学
Evolutionarily Stable Strategy(ESS)	生物进化的稳定策略
Evolutionary Game Theory, EGT	进化对策论
exterior penalty method	外部罚函数法
expected value model	期待值模型
F	
farthest insertion	最远插入法
feasible region	可行域
feasible solution	可行解
Field Programmable Gate Array, FPGA	现场可编程门阵列
Finite State Machine , FSM	有限状态机
Flow shop Scheduling Problem, FSP	物流调度问题
Flexible Scheduling System	柔性调度系统
fine - grained PGA	细粒度并行遗传算法
fitness	适应度
fitness function	适应度函数
fitness scaling	适应度尺度变换
floating - point genes	浮点数编码基因
function optimization	函数优化
Fuzzy Logic, FL	模糊逻辑
fuzzy logic controller	模糊逻辑控制器
G	
GA deceptive problem	GA 欺骗问题
game theory	对策论
gene	基因
generation	世代
generational gap	代沟

generalized crossover model 广义交叉模型
Genetic Algorithm, GA 遗传算法
Genetic Algorithm based graph grammar 图文遗传算法
genetic operators 遗传算子
Genetic Programming, GP 遗传程序设计
genetics 遗传学
GENOCOP 约束优化的遗传算法
genome 基因组
genotype 基因型
global searching 全局搜索
grammar tree 文法树
granular computing 粒度计算
Gray codes 格雷码
Genetic - based machine learning, GBML 基于遗传的机器学习
geometrical crossover 几何交叉
generalized reduced gradient method, GRD 广义简约梯度法
gradient projection method 梯度投影算法
greedy algorithm 贪心算法
group method of data handling, GMDH 数据处理的组方法

H

Hamming distance 海明距离
heterogeneous recombination 异种交叉
heredity 遗传
heuristic crossover 启发式交叉
heuristic method 启发式方法
Hierarchic Genetic algorithm 层次遗传算法
hill - climbing search 爬山法
hybrid genetic algorithm, HGA 混合遗传算法
hybrid soft computing 混合软计算
hybrid system 混合系统

I

image processing 图像处理
image segmentation 图像分割
immune systems 免疫系统
implicit parallelism 隐含并行性
individual 个体
inherent parallelism 内在并行性
initial population 初始种群
intermediate recombination 中间重组
interior penalty method 内部罚函数法
inverse operator 倒位算子
island model 孤岛模型
iterative rule learning approach 规则迭代学习法

J

Job shop Scheduling Problem, JSP 作业调度问题

K

Knapsack problem 背包问题

L

Life - time Fitness Evaluation, LTFE 生命周期适应度评价
linear programming problem 线性规划问题
linear recombination 线性重组
linear scaling 线性尺度变换

local searching 局部搜索
locus 基因座
local selection 局部选择
loss of diversity 多样化损失

M

machine learning 机器学习
machine learning from discovery 机器发现
Markov chain 马尔可夫链
mating 交配
mating rule 交配规则
Michigan approach 密歇根方法
migration 迁移
MIMD 多指令流多数据流
Minimal Deceptive Problem, MDP 最小欺骗问题
multi－modal optimization 多模态最优化
Multi－Objective Optimization, MO 多目标最优化
Multi－objective Genetic Algorithm, MOGA 多目标遗传算法
multiple point crossover 多点交叉
multiplier method 乘子法
mutation 变异
mutation operator 变异算子
mutation probability 变异概率

N

Nagoya approach 名古屋方法
natural selection 自然选择
nearest insertion 最近插入法
nearest neighbor 近邻法
neighbourhood model 邻居模型
neutral theory of molecular evolution 分子进化中性理论
niche 小生境
Niched Genetic Algorithms, NGA 基于小生境技术的遗传算法
niched Pareto genetic algorithm 小生境 Pareto 遗传算法
nonlinear programming problem 非线性规划
Nondeterministic Polynomial Completeness, NP NP 完全

O

objective function 目标函数
off－line performance 离线性能
offspring 子代群体
on－line performance 在线性能
one－point crossover 单点交叉
optimal solution 最优解
optimization 最优化
Order Crossover, OX 顺序交叉
ordinal representation 顺序表示
overspecification 描述过剩

P

panmixis population 随机交配群体
parallel genetic algorithm 并行遗传算法
parallelism 并行性
Partially Matched Crossover, PMX 部分匹配交叉
Pareto optimality Pareto 最优性
Pareto optimal solution Paeto 最优解

path representation	路径表示
pattern recognition	模式识别
penalty function	罚函数
permutation	排列
phenotype	表现型
Pitt approach	匹兹堡方法
primitive functions	原始函数
population	种群
population average fitness	种群平均适应度
population diversity	种群多样性
population size	种群大小
power law scaling	幂尺度变换
predator－prey relations	食物链关系
premature convergence	过早收敛
preference	偏好
preselection	预选择
probabilistic computing	比例模型
Programmable Logical Device, PLD	概率计算
proportional model	可编程逻辑器件
proportional fitness assignment	按比例的适应度计算
Punctuated Equilibrium Evolution	间断平衡进化
Pseudo－Bacterial Genetic Algorithm, PBGA	伪细菌遗传算法
Q	
quadratic programming	二次规划
R	
radical base function Neural Network	径向基神经网络
random algorithms	随机算法
random searching	随机搜索
rank－based fitness assignment	基于排序的适应度计算
rank－based model	排序选择模型
real valued recombination	实值重组
region growing	区域生长
recursive least square method	递归最小二乘法
Reinforcement Learning Algorithm, RLA	增强学习算法
reproduction	复制，再生
ribonucleic acid, RNA	核糖核酸
robustness	健壮性
roulette wheel selection	轮盘赌选择
S	
saltation	跳跃进化
scheduling problem	调度问题
schema	模式
schema defining length	模式定义距
schema order	模式阶
schema theorem	模式定理
selection intensity	选择强度
selection pressure	选择压力
selection variance	选择方差
selection	选择
sharing function	分享函数
shuffle crossover	洗牌交叉
SIMD	单指令单数据流

simulated annealing	模拟退火
Simulated Annealing Genetic Algorithm, SAGA	模拟退火遗传算法
Simple Genetic Algorithm, SGA	基本遗传算法
simulation	模拟
soft computing	软计算
sphere crossover	球面交叉
splice operator	拼接算子
split and merge	分裂合并法
steady state reproduction	稳态繁殖
steady state reproduction without duplicates	没有重串的稳态繁殖
stochastic universal sampling	随机遍历抽样
synthetic theory of evolution	现代综合进化论
steepest descent method	最速下降法
stepping－stone model	踏脚石模型
stochastic tournament model	随机锦标赛模型
symbiosis	共生
symbolic empirical learning	符号经验学习
system identification	系统辨识
systolic & wavefront array	禁忌搜索
T	
Tabu Search, TS	舒张阵列
terminations	终止条件
terminals	终结点
test function	测试函数
The Prisoner's Dilemma	锦标赛选择
tournament selection	囚犯困境
Traveling Salesman Problem, TSP	巡回旅行商问题
truncation selection	截断选择
two－point crossover	两点交叉
U	
underspecification	描述不足
uniform crossover	均匀交叉
uniform mutation	均匀变异
V	
variation	变异
Variants of Canonical Genetic Algorithms, VCGA	遗传算法的多种变形
X	
XOR	异或